Universidade Federal do Pampa
Helison Réus Teixeira

Desenvolvimento de um Módulo de Web Semântica para um Sistema de Suporte à Inspeção de Software

Alegrete
2014

Helison Réus Teixeira

Desenvolvimento de um Módulo de Web Semântica para um Sistema de Suporte à Inspeção de Software

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de software.

Orientador: Prof. Me. João Pablo Silva da Silva

Alegrete

2014

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

T266d Teixeira, Helison Réus
Desenvolvimento de um Módulo de Web Semântica para um Sistema de Suporte à Inspeção de Software / Helison Réus Teixeira.
103 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE SOFTWARE, 2014.
"Orientação: João Pablo Silva da Silva".

1. Web Semântica. 2. Ontologia. 3. Inspeção de Software. 4. Componente. I. Título.

Helison Réus Teixeira

Desenvolvimento de um Módulo de Web Semântica para um Sistema de Suporte à Inspeção de Software

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de software.

Trabalho de Conclusão de Curso defendido e aprovado em 24 de março de 2014

Banca examinadora:

Prof. Me. João Pablo Silva da Silva
Orientador

Prof. Me. Jean Felipe Cheiran
UNIPAMPA

Prof. Me. Sam da Silva Devincenzi
UNIPAMPA

Este trabalho é dedicado ao meu pai,
que apesar de não estar mais presente comigo
estará sempre em minha memória me dando forças.

Agradecimentos

Agradeço a Deus por ter me dado saúde e força para superar as dificuldades.

Ao professor João Pablo Silva da Silva, pela orientação, apoio e confiança.

À minha mãe que se manteve sempre forte e me dando forças para continuar e chegar até aqui.

Ao meu pai por ter sempre acreditado em mim me dando forças para seguir em frente.

Agradeço aos meus irmãos, que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir de constante dedicação no presente!

Agradeço a minha namorada Josieli Breitenbach, que dedicou suas (e nossas) horas sempre me incentivando durante esse cansativo processo que foi o desenvolvimento dessa monografia. Agradeço-a por todos os beijos e puxões de orelha necessários para a continuidade deste trabalho.

Aos meus padrinhos e amigos que sempre me acolheram e me receberam de braços abertos nos poucos momentos de folga que tive.

Ao João Pablo Silva da Silva por ser um orientador sempre presente, esclarecendo minhas dúvidas e me auxiliando quando possível.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado!

"Não é da luz que precisamos, mas do fogo;
não da leve garoa, mas do trovão.
Precisamos da tempestade, do redemoinho e do terremoto.
(Frederick Douglass)

Resumo

Em 1976, Fagan propôs um processo para a inspeções de software. Essa proposta surgiu baseada na necessidade de uma melhoria na qualidade dos produtos de software desenvolvidos naquele tempo. Porém, o processo de inspeção ficou conhecido por sua dificuldade de implantação, pois para poder utilizá-lo muitas empresas o adaptavam, ou utilizavam sistemas que de alguma forma automatizassem a execução do mesmo. Alguns desses sistemas de automatização oferecem serviços que acessam bases de conhecimento, onde o processo de inspeção está representado, permitindo assim a automatização das atividades desse processo. A ontologia é definida como uma base de conhecimento, sendo uma especificação explicita e formal capaz de representar conceitos de domínio através do relacionamento entre esses conceitos, logo através do uso de uma ontologia é possível representar os conceitos envolvidos na inspeção de software. Tendo em vista a necessidade de automatizar o processo de inspeção, foi criado um sistema de suporte à inspeção de software que faz parte de um projeto de pesquisa. O trabalho aqui apresentado faz parte desse projeto de pesquisa e tem por objetivo o desenvolvimento de um módulo de Web Semântica para o sistema de suporte à inspeção baseado na IEEE 1028. Esse módulo de Web Semântica provém serviços que permitem gerenciar ontologias previamente carregadas, informando indivíduos e suas propriedades à mesma. Também são providos serviços para remoção, atualização, e recuperação dos indivíduos da ontologia. O acesso aos serviços do módulo é feito feito através de interfaces de comunicação disponibilizadas por ele. Para a criação desse componente foi utilizada a OWL API, através dela é possível manusear os indivíduos da ontologia e seus relacionamentos. Após o desenvolvimento do módulo de Web Semântica o mesmo foi validado integrado-o ao sistema de suporte à inspeção de software. Utilizando os serviços do módulo de Web Semântica, foi possível popular uma ontologia utilizada pelo sistema de suporte e utilizar o serviço de consultas oferecido pelo módulo para recuperar as informações necessárias para a automatização de algumas fases do processo de inspeção de software. Apesar do módulo ter sido desenvolvido para trabalhar com o sistema de suporte à inspeção de software ele é capaz de trabalhar com qualquer sistema que necessite de seus serviços, pois os serviços oferecidos pelo módulo são genéricos para qualquer ontologia.

Palavras-chave: Web Semântica. Ontologia. Inspeção de Software. Componente.

Abstract

In 1976, Fagan proposed a process for software inspections, this proposal was based on the need for an improvement in the quality of software products developed at that time. However, due to its systematic, he became known for being difficult to implant, because was necessary adapt it to be able to be used, or just use other systems that somehow either automated the execution of software inspections. Some of these automated systems provide services that access knowledge bases, which represent the software inspection process, allowing the automation of activities in this process. Ontology is an explicit and formal specification able to represent domain concepts through the relationship between these concepts, and then through the use of an ontology can represent concepts involving software inspection. Based on the need to automate the inspection process, it was created an inspection support system software, which is part of a research project. The work presented here is part of this research project, this work has for objective the development of a Semantic Web module for software inspection system based on IEEE 1028. This Semantic Web module provides services that enable management of ontologies previously loaded by the module, informing individuals and their properties to it. Services are also provided for deletion, updating, and retrieval of individuals of the ontology. For that, this semantic Web module was developed in as a component, which can be used by any system that requires the use of ontologies. This component can be accessed through communication interfaces provided by the module, which provide the same services to any system. For the creation of this component has been used the OWL API, through it you can handle the ontology individuals and their relationships. To retrieve individuals were created methods based on their relationships, and methods that performing SPARQL queries can retrieve such individuals. After the development of the Semantic Web module it was integrated to the software inspection support system. Using the services of the Semantic Web module, which worked with an ontology that represents the software inspection process was possible to populate the ontology with his individuals, and use the SPARQL query service offered by the module to retrieve the information necessary for the automation of some phases of the software inspection process. Although the module has been developed to work with the software inspection support system, but it is able to work with any system requiring his services, to it is necessary to load the ontology that will be used.

Key-words: Semantic Web. Ontology. Software Inspection. Component.

Lista de ilustrações

Lista de siglas

API *Application Programming Interface*

CMMI *Capability Maturity Model Integration*

EL *Existencial Logic*

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos

IRI *Internationalized Resource Identifier*

OWL 2 *Ontology Web Language 2*

PPQA *Process and Product Quality Assurance*

QL *Query Language*

RDF *Resource Description Framework*

RDF-S *Resource Description Framework Schema*

RIF *Rule Interchange Format*

RL *Rule Language*

SKOS *Simple Knowledge Organization System*

SPARQL *SPARQL Protocol and RDF Query Language*

SWRL *Semantic Web Rule Language*

TTCN-3 *Testing and Control Notation - 3*

URI *Uniform Resource Indentifier*

W3C *World Wide Web Consortium*

XML *Extensible Markup Language*

Sumário

1 Introdução

1.1 Contexto

Em 1976, Fagan percebeu a necessidade da criação de um processo para garantir a qualidade do software que era desenvolvido, sendo assim ele propôs um processo de inspeção com o objetivo de encontrar possíveis erros durante o ciclo de desenvolvimento do software (FAGAN, 1999). Desde então, para garantir a qualidade de seus produtos, empresas de software têm utilizado a inspeção (AURUM; PETERSSON; WOHLIN, 2002).

Existem revisões sistemáticas e não sistemáticas. Ambas são realizadas em artefatos de software, um artefato de software segundo a IEEE (2008) é um conjunto de programas, procedimentos, dados e documentos gerados durante a criação de um software. As revisões de software têm como objetivo encontrar não conformidades em documentos, processos e artefatos gerados durante o processo de desenvolvimento do software. A inspeção é classificada como uma revisão sistemática.

As revisões não sistemáticas ocorrem de forma natural, um exemplo seria uma discussão sobre algum problema técnico durante a pausa para o café dos membros de uma empresa (PAULA FILHO, 2009), já as revisões sistemáticas são revisões nas quais existe a participação de uma equipe de revisores, essa equipe, baseia-se em procedimentos para conduzir a revisão, após realizar a revisão é gerado documentos com os resultados encontrados (IEEE, 2008).

A inspeção de software tem como objetivo identificar anomalias em produtos de software é considerada uma anomalia qualquer condição que não esteja de acordo com os requisitos do software, documentos de projeto e de usuário, desvios de padrões usados e especificações técnicas (IEEE, 2008).

As inspeções de software são realizadas por equipes de inspeção. Essas equipes são formadas por pessoas capacitadas que conhecem a engenharia de software (PAULA FILHO, 2009). A efetividade de uma inspeção pode estar relacionada com a sua condução (PAULA FILHO, 2009), porém o sucesso ou fracasso de uma inspeção está relacionado ao conhecimento dos inspetores envolvidos e à cobertura dessas inspeções.

Fagan, com seus estudos, provou que inspeções podem detectar de 50 a 92 por cento de defeitos durante o processo de criação do software (FAGAN, 1999). Essa detecção se deve ao fato da inspeção de software ser utilizável durante todo o processo de desenvolvimento, logo o uso de inspeções garante a entrega de um melhor produto, reduzindo assim a manutenção necessária nos mesmos.

Os sistemas de suporte à inspeção de software aumentam a cobertura e facilitam o controle das inspeções, permitindo que algumas atividades de inspeção de software como a entrega dos artefatos (trabalhos a serem inspecionados) pelo autor possa ser realizada pelo sistema, assim como, os resultados das inspeções podem ser armazenados no sistema, facilitando o acesso por parte de todos os envolvidos no processo de inspeção. (HEDBERG; LAPPALAINEN, 2005)

A Web Semântica representa computacionalmente o conhecimento envolvido em determinados domínios, tratando informações como uma rede de conceitos, esses conceitos estão ligados por relacionamentos, permitindo assim dar significado aos conteúdos por ela representados (LIBRELOTTO; RAMALHO; HENRIQUES, 2003). Através do uso de tecnologias relacionadas a Web Semântica é possível provê-la (GRUBER, 1995). Dentre essas tecnologias as ontologias se destacam como forma de representar esse conhecimento.

Um processo é criado para executar atividades de um determinado domínio, esse domínio pode ser representado através de uma ontologia. Através dessa ontologia é possível obter o conhecimento envolvido na execução do processo, permitindo assim que esse conhecimento seja utilizado por sistemas computacionais (CHANDRASEKARAN et al., 1999).

A inspeção de software é aplicada através do processo de inspeção, para isso é formado um time de inspeção composto normalmente de pessoas envolvidas em todas as fases do processo do software (programadores, testadores, analistas, projetistas ...) devido a isso essas pessoas acabam exercendo outras funções, aumentando assim o esforço aplicado por elas durante a execução do processo de software. Esse esforço pode ser reduzido através do uso de um sistema de suporte à inspeção de software (DENGER; SHULL, 2007).

Esse trabalho faz parte de um projeto de pesquisa que visa diminuir o esforço aplicado durante a inspeção de software, aumentando assim a cobertura da inspeção. Para isso, através do projeto de pesquisa foi criado um Sistema de Suporte à Inspeção de Software. Esse sistema foi projetado e desenvolvido em módulos. Esses módulos são responsáveis por diferentes serviços.

O trabalho aqui apresentado corresponde a um desses módulos, que seria o módulo de Web Semântica. Esse módulo trabalha com ontologias para manter parte do conhecimento existente no domínio do processo de inspeção de software.

O uso do módulo de Web Semântica não está restrito somente ao Sistema de Suporte de Inspeção de Software, pois o mesmo foi desenvolvido e projetado para trabalhar com qualquer ontologia.

1.2 Motivação

Apesar dos benefícios da inspeção de software as empresas não conseguem visualizar uma relação entre as inspeções e a qualidade do produto. Essas empresas adaptam processos de inspeção externos, e esses acabam sendo incompatíveis com a experiência dos membros da equipe ou requerem um esforço indisponível. Outras empresas consideram que a inspeção de software requer um investimento inicial muito alto, pois os envolvidas no processo necessitam deixar suas atividades principais de lado para realizar a inspeção, e para outras, isso além de ser muito caro, acaba gerando uma perda na produtividade, pois esses recursos não estão alocados em suas funções principais (DENGER; SHULL, 2007). Essas empresas não conseguem levar em consideração o fato de que uma inspeção de software bem aplicada aperfeiçoa a qualidade do produto e gera melhorias no controle do processo e no gerenciamento do projeto de software (FAGAN, 1999).

A motivação desse trabalho está na criação de um módulo de Web Semântica que faça a integração de diversas tecnologias existentes a fim de facilitar o uso da Web Semântica. Com essa integração o usuário do módulo poderá acessar os serviços da Web Semântica sem a necessidade de estudar as *Application Programming Interfaces* (APIs) e *frameworks* disponíveis para trabalhar com ontologias, pois o módulo engloba tais tecnologias.

Este módulo, utilizado pelo Sistema de Suporte à Inspeção de Software é capaz de recuperar o conhecimento envolvido no domínio do processo de inspeção (representado em uma ontologia carregada pelo módulo), provendo o conhecimento necessário ao sistema de suporte para que o mesmo automatize algumas atividades do processo de inspeção de software.

1.3 Objetivo Geral e Específico

Esse trabalho tem por objetivo o desenvolvimento de um módulo de Web Semântica em forma de componente para prover serviços de consulta e manipulação do conhecimento existente em uma ontologia. Para o desenvolvimento desse módulo de Web Semântica os seguintes objetivos específicos são propostos:

- Prover através do módulo de Web Semântica serviços que permitam a manipulação dos indivíduos de uma ontologia;
- Prover através do módulo de Web Semântica serviços que permitem a consulta dos indivíduos de uma ontologia;
- Projetar e desenvolver o módulo de Web Semântica em forma de um componente;

- Realizar a integração do módulo de Web Semântica com o sistema de suporte à inspeção de software;

1.4 Principais Contribuições

As principais contribuições desse módulo são:

- criação de um módulo de Web Semântica na forma de um componente que provê interfaces, permitindo assim a substituição do mesmo;
- integração de tecnologias que trabalham com ontologias, permitindo o acesso e manipulação dessas ontologias e seus indivíduos através das interfaces do módulo;

1.5 Organização do Trabalho

A organização desse trabalho pode ser vista abaixo, onde os próximos capítulos são apresentados com uma breve descrição do que é apresentado em cada capítulo.

- **Capítulo 2**: nesse capítulo é apresentado a fundamentação teórica e tecnológica necessária para o desenvolvimento do módulo de Web Semântica aqui proposto.
- **Capítulo 3**: nesse capítulo são apresentados e analisados os trabalhos relacionados ao trabalho aqui proposto.
- **Capítulo 4**: nesse capítulo é apresentado o problema a ser solucionado nesse trabalho juntamente com a sua solução.
- **Capítulo 5**: nesse capítulo são apresentados os testes realizados para validar o módulo de Web Semântica.
- **Capítulo 6**: nesse capítulo é apresentado as considerações parciais desse trabalho juntamente com o cronograma de atividades a serem realizadas para concluir o trabalho aqui proposto.

2 Fundamentação Teórica-Tecnológica

Neste capítulo é apresentado a fundamentação teórica que serve como base de introdução aos conceitos e tecnologias que utilizadas durante o desenvolvimento do trabalho aqui apresentado. Na seção 2.1 é abordado o processo de inspeções de software, em seguida, na seção 2.2 a Web Semântica é apresentada. Na seção 2.3 a ontologia é abordada, nessa seção também é apresentado a *Ontology Web Language 2* (OWL 2), o *SPARQL Protocol and RDF Query Language* (SPARQL) e o *Semantic Web Rule Language* (SWRL). Na seção 2.4 é apresentada a engenharia de componentes, a qual foi utilizada para a elaboração desse módulo. Na seção 2.5 são abordadas as APIs utilizadas para o desenvolvimento do módulo aqui apresentado, que seriam a OWL API, utilizada para gerenciar a ontologia e o pacote de consultas *Jena Query* responsável pela execução das consultas SPARQL. Por fim na seção 2.6 é apresentado o fechamento desse capítulo.

2.1 Inspeção de Software

O Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) apresenta a IEEE 1028, que é um conjunto de padrões para a realização de revisões de software sistemáticas, dentre esses padrões, se encontra a inspeção de software. Nesse contexto, revisão sistemática são revisões onde existe a participação de um time, seus resultados e procedimentos são documentados (IEEE, 2008).

Hoje podemos dizer que a inspeção é um importante meio para a verificação da qualidade do software. A inspeção possui entre seus benefícios o fato dela poder ser aplicada em qualquer artefato de software produzido durante a fase de desenvolvimento. É provável que seja um dos métodos atualmente mais aceitos para melhorar a qualidade dos produtos, porém, nem sempre é aplicado (AURUM; PETERSSON; WOHLIN, 2002).

Para aplicar a inspeção é necessário um time de inspeção com papéis específicos, que seriam (IEEE, 2008):

- líder da inspeção: conduz a inspeção durante suas fases;
- registrador: registra as anomalias detectadas durante a reunião de inspeção;
- leitor: ajuda a compreender e faz a leitura dos artefatos a serem inspecionados;
- autor: responsável pelo artefato a ser inspecionado;
- inspetor: inspeciona os artefatos.

Para uma inspeção ocorrer é necessário no mínimo dois participantes, o autor do artefato de software e o líder de inspeção. Nesse caso o líder da inspeção além de líder terá os papéis de inspetor e registrador, e o autor do artefato também será o leitor, pois o autor não pode inspecionar um artefato que o mesmo produziu. (IEEE, 2008)

Ao final de cada inspeção uma lista de anomalias classificadas e organizadas é gerada. O processo de inspeção também gera outros dados que devem ser utilizados por especialistas em garantia da qualidade do processo, verificando possíveis melhorias a serem realizadas no processo (IEEE, 2008).

Com base nessas classificações e organizações é possível verificar quais atividades possuem problemas e quais devem ser melhoradas. Essas avaliações devem ocorrer regularmente, e o processo de inspeção deve ser adaptado continuamente, garantindo assim uma maior efetividade do processo (IEEE, 2008).

2.1.1 Processo da Inspeção

Recomenda-se que os seguintes itens já tenham sido preparados para poder inciar uma inspeção: homologação dos objetivos da inspeção; artefato de software entregue ao inspetor; documentos contendo os procedimentos da inspeção; formulários para se reportar os resultados das inspeções; lista de anomalias ou uma lista de problemas possíveis; e os requisitos do sistema, ou outros documentos de especificação. Além desses itens, recomenda-se que outros itens já tenham sido preparados, porém esses não são obrigatórios. Eles seriam: *checklists* da inspeção; critérios de qualidade necessários para uma reinspeção; lista de artefatos que possuam relação com o a ser inspecionado; padrões, regulamentações, diretrizes, especificações e procedimentos que serão necessários para inspecionar o artefato; especificações de hardware, software e instrumentos necessários para o artefato; dados de performance; e categorias de possíveis anomalias (IEEE, 2008).

Após a verificação e disponibilização desses itens é necessário verificar se o artefato está de acordo com os critérios de entrada. Um desses critérios para se dar entrada na inspeção é a verificação do planejamento do projeto, pois a inspeção deve estar documentada no projeto, e caso alguma inspeção extra seja necessária ela deve ser requisitada ou pelo gerente do projeto, ou o gerente de qualidade, ou o autor. Esse processo de requisição deve ser feito seguindo os procedimentos locais (IEEE, 2008).

Para que uma inspeção ocorra, é necessário que alguns eventos tenham ocorrido, que seriam: o líder da inspeção deve informar que o artefato está pronto para ser inspecionado; uma ferramenta de detecção de erros automática deve ter sido utilizada para detectar menores erros; marcos anteriores dos quais o artefato é necessário já terem sido inspecionados e terem tido um resultado satisfatório; e para uma reinspeção é necessário que todas as anomalias identificadas anteriormente tenham sido resolvidas (IEEE, 2008).

Além dos critérios de entrada da inspeção existem os critérios de saída, que devem ser atendidos para que a inspeção seja encerrada. Esses critérios incluem a aprovação por parte do líder da inspeção e uma lista de evidencias que são geradas durante o processo de inspeção (IEEE, 2008).

A inspeção de software pode ser dividida em sete fases, essas são apresentadas subsequentemente.

2.1.1.1 Preparação Gerencial

Antes de se inciar o planejamento da inspeção é necessário que seja realizado a preparação do gerenciamento, onde os gerentes devem garantir que a inspeção seja realizada conforme as normas e procedimentos padrões, e que elas sigam leis aplicáveis, contratos e politicas locais (IEEE, 2008). Para esse fim a IEEE 1028 determina que os gerentes: planejem o tempo e os recursos para se realizar a inspeção, incluindo o time de inspeção, financiem a inspeção, a infraestrutura e as facilidades necessárias para planejar, definir, executar e gerenciar a inspeção; disponibilizem treinamento e orientação para os inspetores; garantam que os membros do time de inspeção possuem a capacidade necessária para realizar a inspeção; garantam a condução das inspeções e a ação sobre as recomendações do time em tempo hábil (IEEE, 2008).

2.1.1.2 Planejamento da Inspeção

Após a preparação do gerenciamento se inicia o planejamento da inspeção, onde autor entrega ao líder da inspeção o artefato a ser inspecionado juntamente com os seus documentos de especificação, para que a inspeção seja planejada (IEEE, 2008).

Durante o planejamento, o líder da inspeção é responsável por montar os times de inspeção, esse time é montado de acordo com a capacidade de cada membro de inspecionar o artefato. O líder da inspeção também informa aos inspetores a duração da reunião e repassa ao time de inspeção o material necessário para que os inspetores possam se preparar para a inspeção (IEEE, 2008).

2.1.1.3 Apresentação dos Procedimentos da Inspeção

Durante a apresentação dos procedimentos da inspeção o inspetor líder responde as dúvidas dos inspetores referentes as ferramentas de avaliação da inspeção assim como as responsabilidades de cada membro da inspeção. Juntamente a isso o inspetor líder deve apresentar o material necessário para a inspeção em tempo suficiente para que o mesmo seja inspecionado pelos inspetores (IEEE, 2008).

2.1.1.4 Apresentação dos Produto de Inspeção

Durante a fase de apresentação do produto da inspeção o autor apresenta o artefato a ser inspecionado explicando aos inspetores os detalhes do artefato. Com base nessa explicação os inspetores poderão fazer suas inspeções individuais durante a fase de preparação. É interessante que esse procedimento seja assistido por todos os envolvidos no processo de desenvolvimento do produto, pois assim todos saberão das principais características daquele artefato a ser inspecionado, evitando possíveis discordâncias relacionadas a ele (IEEE, 2008).

2.1.1.5 Preparação

Durante a preparação para a inspeção os membros do time de inspeção inspecionam individualmente o artefato de software que lhe foi passado. Nessa inspeção as anomalias encontradas são documentadas para serem discutidas posteriormente na reunião de inspeção. Após essa inspeção individual o documento gerado pelos inspetores é entregue ao líder da inspeção, onde ele categoriza as anomalias e verifica a necessidade da reunião. Caso o numero de anomalias seja muito grande, e os critérios de entrada não sejam atingidos, o líder cancela a inspeção e repassa o artefato para o autor, e então o autor deve reparar tais anomalias para então se reiniciar o processo de inspeção (IEEE, 2008).

Durante a preparação também é importante que seja definido uma forma em que os artefatos serão lidos durante a reunião de inspeção, essa decisão pode ser tomada tanto pelo líder da inspeção, quanto pelo leitor (IEEE, 2008).

2.1.1.6 Reunião de Inspeção

Durante a reunião de inspeção time de inspeção é apresentado. O leitor da inspeção explica a todos presentes quais artefatos estão sendo inspecionados e tira duvidas sobre eles, caso o leitor não seja capaz disso, o autor deve fazer essa explicação. Após isso o leitor apresenta a lista de anomalias já classificados pelo líder da inspeção e elas são discutidas juntamente com o autor verificando se o que foi encontrado realmente é uma anomalia. Após essa discussão os resultados são anotados pelo registrador e repassados para o líder da inspeção, então, inicia-se a revisão do produto (IEEE, 2008).

Durante a revisão do produto o time de inspeção inspeciona em conjunto o artefato de software em busca de alguma anomalia que não havia sido identificada. Caso algo seja encontrado, é feita uma discussão sobre essa anomalia e o registrador atualiza o documento de anomalias (IEEE, 2008).

No final da reunião de inspeção, o documento contendo as anomalias encontradas é lido, e caso haja alguma discordância entre o time de inspeção ou com o autor, o líder

da inspeção deve permitir que a anomalia em questão seja discutida. Durante essa fase é muito importante que o líder da inspeção mantenha o foco do time na detecção da anomalia, não em possíveis soluções. Como a reunião tem seu tempo limitado, conflitos que não são resolvidos rapidamente são registrados juntamente com a anomalia (IEEE, 2008).

2.1.1.7 Retrabalho / Continuidade

Após a reunião de inspeção a equipe de inspeção deve verificar a lista de anomalias e decidir o que será feito com o artefato de software. Existem três possíveis decisões a serem tomadas (IEEE, 2008):

- *Aceitar o artefato sem reinspeção:* o artefato é aceito sem a necessidade de se realizar uma nova inspeção.
- *Aceitar o artefato com verificação do retrabalho:* o artefato é aceito somente depois que alguém do time de inspeção, ou o líder faça uma verificação individual no artefato (sem necessidade de reunião).
- *Reinspeção:* o artefato só será aceito após ser inspecionado novamente.

2.2 Web Semântica

Através das páginas da Web podemos compartilhar informações com pessoas em diferentes lugares do mundo, essas páginas são compostas de dados. Muitas dessas páginas Web possuem *links* que nos levam a outras páginas, onde algo relacionado ao assunto é apresentado. Nós conseguimos enxergar essas referências e compreender essas informações, porém, em alguns casos, as máquinas ainda não conseguem compreender e manipular esses dados, elas não identificam a relação entre os dados que estão sendo apresentados. Para permitir que isso seja possível foi proposto a Web Semântica (BERNERS-LEE et al., 2001).

A Web Semântica possui uma estrutura necessária para permitir que os computadores possam manusear o conteúdo da Web, criando um ambiente de conteúdos ligados que permitem aos agentes de software percorrer páginas da Web executando tarefas complexas para os usuários (BERNERS-LEE et al., 2001). Além disso a Web Semântica é capaz de tornar a internet inteligente. Para isso é necessário que as páginas Web possuam uma semântica clara e definida, permitindo o raciocínio ontológico por parte de agentes (FREITAS, 2003).

Para a Web Semântica ser aplicada não é necessário a criação de uma nova Web, pois a Web Semântica é uma extensão da Web atual. Para torná-la realidade é necessário

que as páginas Web sejam compreensíveis pelas máquinas através das tecnologias já existentes (BERNERS-LEE et al., 2001). Para isso a *World Wide Web Consortium* (W3C) criou um padrão para se trabalhar com a Web Semântica. Esse padrão permite que pessoas criem bancos de dados Web e escrevam regras para manusear os dados, permitindo assim conectar os dados das páginas Web. Esses dados conectados são qualificados por tecnologias como o *Resource Description Framework* (RDF) , SPARQL, OWL e *Simple Knowledge Organization System* (SKOS) (W3C, 2013).

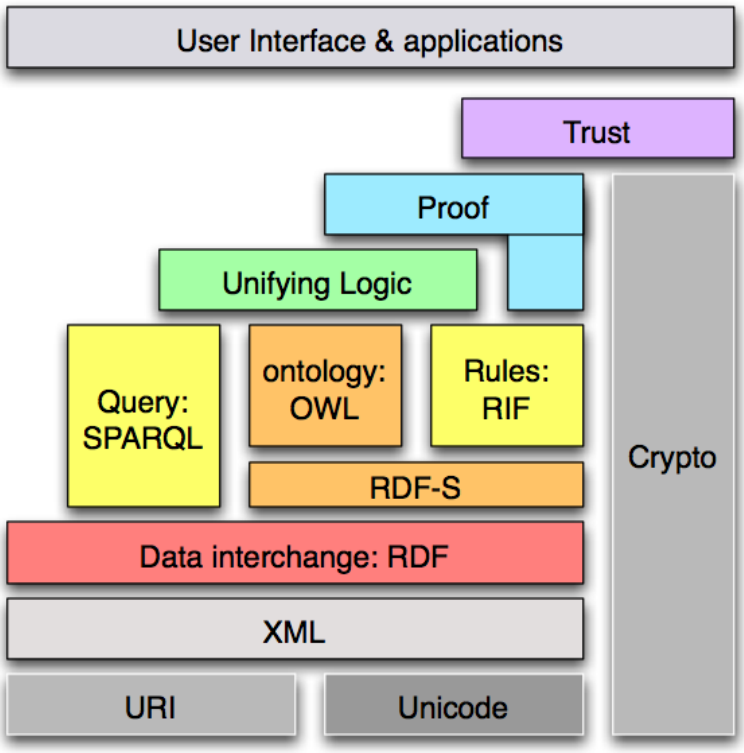
A arquitetura da Web Semântica é representada através de uma pilha, conforme a Figura 1. Nessa pilha podemos identificar que a arquitetura da Web Semântica está dividida em camadas, cada camada é responsável por determinada função, conforme pode ser visto a seguir: (WAGNER FILHO; LÓSCIO,)

- *Uniform Resource Indentifier (URI) e Internationalized Resource Identifier (IRI):* permite identificar os recursos da Web Semântica;
- *unicode:* responsável por manter a consistência da escrita;
- *Extensible Markup Language (XML) :* permite a criação de marcadores;
- *RDF:* disponibiliza o modelo de descrição lógica dos dados;
- *Resource Description Framework Schema (RDF-S)* provê o vocabulário necessário à camada RDF;
- *ontology (OWL):* estende a camada RDF aumentando sua expressividade;
- *SPARQL:* permite consultas à OWL;
- *Rule Interchange Format (RIF) :* responsável pelas regras da ontologia;
- *unifying logic:* responsável pelo raciocínio e execução das inferências;
- *proof:* avalia o nível de confiabilidade dos recursos e informações;
- *user interface & applications:* camada de aplicações que interagem com a Web Semântica.

Acredita-se que com o aumento do uso da Web Semântica, a pesquisa na Web se torna mais produtiva, pois com sua capacidade de ligação dos dados ela os seus resultados são mais relevantes. Atualmente é difícil encontrar essas informações, devido a falta de semântica associada entre os recursos de informação (FALBO et al., 2004).

No contexto de Web Semântica as informações são tratadas como uma rede de conceitos. O objetivo disso é dar significado aos conteúdos da Web através de dados que

Figura 1: Pilha da Web Semântica representando sua arquitetura



Fonte: (BARNERS-LEE, 2006)

possam ser processados por máquinas. Esses conceitos podem estar relacionados com grupos de informações ou entre eles (LIBRELOTTO; RAMALHO; HENRIQUES, 2003).

Uma das formas de se dar significado aos conteúdos da Web e relacionar conceitos de informação é o uso de ontologias, que será visto na seção 2.3.

2.3 Ontologia

Na filosofia a ontologia é vista como uma disciplina que estuda teorias sobre a natureza da existência das coisas (GRUBER, 1995). Na computação a ontologia é entendida como a especificação de uma conceitualização, ou como um conjunto de termos hierárquicos que descrevem um domínio que serve como base de conhecimento. Logo, formalmente, uma ontologia é a declaração de uma teoria lógica (GRUBER, 1995).

A ontologia torna possível representar conhecimentos através de um formalismo declarativo, para isso é necessário descrever um conjunto de objetos que se relacionam e são capazes de expressar o significado de algum domínio. Esse relacionamento deve ser refletido em um vocabulário representacional no qual o sistema que irá trabalhar com ele o possa compreender (GRUBER, 1995).

2.3.1 Web Ontology Language 2 - OWL 2

Através do uso de *Resource Description Framework* (RDF) é possível representar informações na Web processáveis por máquinas, permitindo assim a automação de atividades através do uso de agentes (KLYNE; CARROLL; MCBRIDE, 2004). Porém, o *World Wide Web Consortium* (W3C) identificou que o RDF não é capaz de representar algumas informações mais expressivas (ANTONIOU; HARMELEN, 2004). Outros grupos de pesquisa tanto na América quanto na Europa já haviam identificado essa necessidade e se uniram para desenvolver uma linguagem mais rica em representação do conhecimento, essa foi chamada de DAML+OIL (ANTONIOU; HARMELEN, 2004).

Com base na estrutura do DAML+OIL e na do RDF, o W3C projetou a *Web Ontology Language* (OWL) que é uma linguagem que permite maior expressividade nas representações do conhecimento, permitindo que as máquinas possam processar o conteúdo de informações ao invés de apenas apresentá-las (ANTONIOU; HARMELEN, 2004). Através da OWL é possível representar conceitos e seus termos, e o relacionamento entre esses termos (MCGUINNESS; HARMELEN, 2004).

Segundo W3C Owl Working Group et al. (2009), a OWL 2 possui as mesmas características da OWL, porém com um maior poder de expressividade, incluindo:

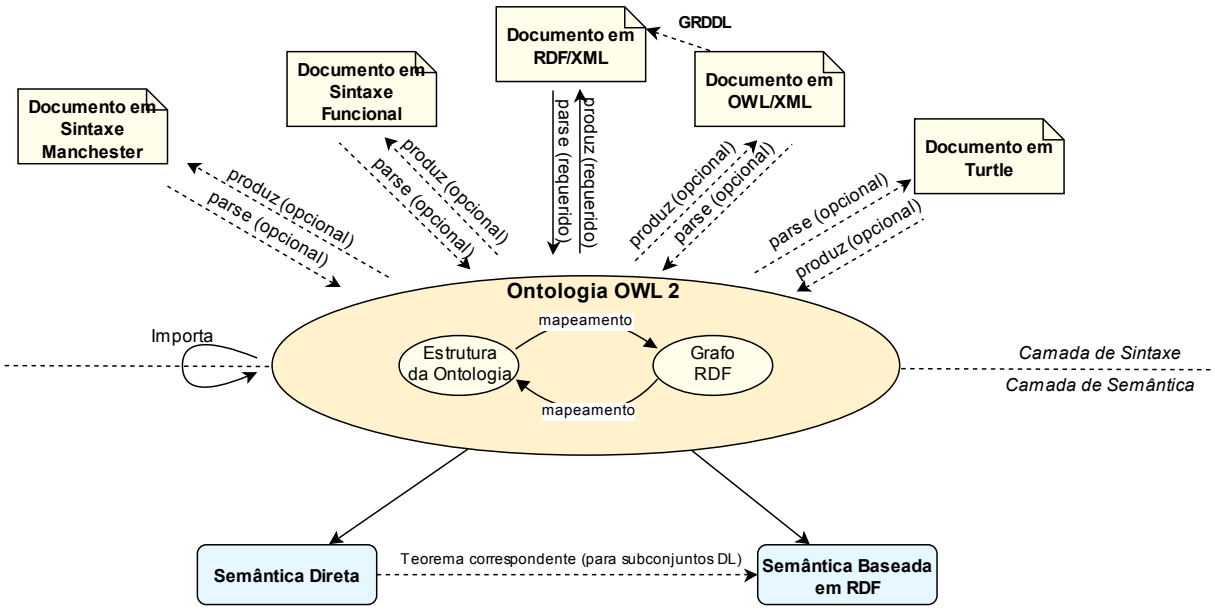
- uso de chaves;
- propriedades tipo dado e objeto em cadeia;
- tipos de dados mais ricos possuindo novos tipos de dados
- propriedades reflexivas, assimétricas e disjuntas;
- anotações mais aprimoradas.

Abaixo podemos ver na Figura 2 uma visão geral da OWL 2, mostrando suas bases tecnológicas e como elas se relacionam.

A OWL 2 define três sublinguagens que a compõe, que seriam a EL (linguagem de descrição), a QL (linguagem de consulta) e a RL (linguagem de regras). Essas são definidas através de restrições sintáticas na estrutura da ontologia. Essa restrições podem ser especificadas através de modificações na gramática funcional da sintaxe, e possivelmente aumentando as restrições globais (MOTIK et al., 2009). As sublinguagens da OWL 2 são descritas logo abaixo:

1. *OWL2 Existencial Logic (EL):* desenvolvida para trabalhar com aplicações que trabalham com muitas propriedades, restrições e classes. Existem alguns algoritmos de raciocínio dedicado para esta sublinguagem que permitem implementações bem flexíveis.

Figura 2: Visão geral da OWL 2 e relacionamento de suas tecnologias bases.



Fonte: W3C Owl Working Group et al. (2009)

2. *OWL2 Query Language (QL):* desenvolvida para trabalhar com aplicativos que utilizam muitas instancias de dados, onde as consultas são mais importantes que o raciocínio. Possui um poder expressivo limitado, porém inclui funcionalidades de modelos conceituais como diagramas de classe UML e diagramas ER.

3. *OWL2 Rule Language (RL):* desenvolvida para trabalhar com aplicações que requerem um raciocínio escalável sem sacrificar o poder da expressividade. Modelada para trabalhar com aplicações em OWL 2 que são capazes de utilizar toda a expressividade da OWL para eficiencia, assim como as aplicações RDF que necessitam de expressividade adicional (MOTIK et al., 2009).

Existem algumas sublinguagens que estendem a OWL 2 QL. Dentre elas podemos destacar a OWL Lite e a OWL DL. A OWL Lite permite o uso de hierarquia e restrições simples, permitindo apenas os valores 0 ou 1 em suas restrições, e a OWL DL permite uma expressividade maior, permitindo o uso de restrições mais complexas, porém essas restrições devem ser computáveis e conclusivas.

As ontologias de qualquer uma das sublinguagens acima podem ser escritas em um documento ontológico usando qualquer uma das sintaxes da OWL 2.

2.3.2 SPARQL Protocol and RDF Query Language (SPARQL)

Proposto em 2004 pela W3C, o SPARQL permite a realização de consultas em modelos de dados RDF (PÉREZ; ARENAS; GUTIERREZ, 2006). Os arquivos OWL como visto anteriormente são codificados em XML/RDF (HORROCKS; PATEL-SCHNEIDER, 2003), logo a consulta dos dados armazenados por esses arquivos deve ser realizada através do SPARQL (KOLLIA; GLIMM; HORROCKS, 2011).

Para se realizar uma consulta através do SPARQL é necessário conhecer sua sintaxe mais profundamente, pois as cláusulas de consulta SPARQL são semelhante as do SQL, porém possuem suas particularidades. Uma consulta simples em SPARQL é composta de duas partes, a cláusula SELECT onde é informado a variável que irá aparecer no resultado da consulta e a cláusula WHERE onde é informado o identificador uniforme do recurso (URI) onde será realizada a consulta juntamente com o termo a ser procurado (PRUD'HOMMEAUX; SEABORNE, 2008). As principais cláusulas utilizadas para a realização de consultas SPARQL são (PRUD'HOMMEAUX; SEABORNE, 2008):

- *SELECT:* retorna as variáveis passadas por ela juntamente com suas correspondências;
- *CONSTRUCT:* retorna um grafo RDF baseado no modelo passado na consulta;
- *ASK:* retorna um booleano informando se encontrou ou não o termo especificado;
- *PREFIX:* utilizado para indicar a ontologia utilizada na consulta;
- *WHERE:* utilizado para definir as condições para a realização da consulta;
- *FROM :* utilizado para identificar a fonte de dados padrão para realizar a consulta;
- *FILTER:* restringe a consulta, permitindo somente os dados onde a condição especificada é verdadeira;
- *OPTIONAL:* semelhante ao filtro, porém o campo passado no OPTIONAL é retornado em branco caso não seja encontrado;
- *UNION:* utilizado para consultar um mesmo termo em diferentes fontes de dados;
- *ORDER BY:* utilizado para ordenar os dados resultados da consulta.

Em seguida é apresentado um exemplo do uso do SPARQL onde são recuperados dois valores da base de dados da ontologia.

Na Figura 3 são apresentados os dados que serão utilizados na consulta.

Na Figura 4 é apresentado a consulta que é realizada, onde é indicado através do prefixo a URI onde os dados serão procurados, em seguida na cláusula SELECT é passado

Figura 3: Dados da ontologia que serão utilizados na consulta

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
```

Fonte: Prud'hommeaux e Seaborne (2008)

duas variáveis onde será armazenado o resultado da consulta, e na cláusula WHERE é informado o que será selecionado da base de dados e em qual variável será armazenado o resultado.

Figura 4: Select e suas cláusulas utilizadas para recuperar os dados da ontologia

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

Fonte: Prud'hommeaux e Seaborne (2008)

Na Figura 5 é apresentado o resultado dessa seleção em formato de tabela, onde os atributos onde foram armazenados os resultados estão nos cabeçalhos da tabela com seus resultados logo abaixo.

Figura 5: Resultado da consulta a base de dados da ontologia

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Fonte: Prud'hommeaux e Seaborne (2008)

2.3.3 Semantic Web Rule Language (SWRL)

A SWRL é composta da combinação entre a OWL DL e OWL Lite com o *Unary/-Binary Datalog RuleML*, possuindo a mesma sintaxe da OWL. Essa linguagem de regras

amplia o conjunto de axiomas da OWL permitindo a combinação de cláusulas de Horn com a base de conhecimento da OWL (HORROCKS et al., 2004). Uma cláusula de Horn é uma cláusula que possui no mínimo um literal positivo (HORN, 1951). Através da SWRL é possível adicionar regras a ontologia que não estão no modelo. Essas estão na forma de implicação entre antecedente e o consequente, fazendo com que o comportamento do antecedente tenha alguma ligação com o do consequente (HORROCKS et al., 2004).

A sintaxe do OWL é composta por uma sequencia de axiomas e fatos, e como a SWRL possui uma sintaxe semelhante a da OWL, ela deve ser estendida com axiomas de regras, seguindo o modelo:

axioma `::=` ***regra***

Onde um axioma de regra composto de um antecedente e de um consequente, onde cada um consiste de um conjunto atômico que pode ser vazio. Um axioma de regra também pode estar associado a referencia de uma URI, que pode servir para identificar a regra como visto no seguinte exemplo (HORROCKS et al., 2004):

regra `::= 'Implica (' [` ***URIReferente*** `]` ***anotação antec conseq*** `')'`
antec `::= 'Antecedente ('` ***átomo*** `')'`
conseq `::= 'Consequente ('` ***átomo*** `')'`

Para facilitar a compreensão da da SWRL, Horrocks et al. (2004) apresentam uma sintaxe informal da SWRL. Nessa, as regras possuem a formula: *antecedente* → *consequente*. Para se indicar as variáveis se utiliza o prefixo padrão da interrogação. Com base nessa sintaxe, pode-se afirmar de que por estar classificado como um estudante, aquele indivíduo também é uma pessoa.

Estudante (?x1) → Pessoa (?x1).

As regras SWRL também podem ser aplicadas diretamente a indivíduos da OWL (O'CONNOR et al.,), como podemos ver abaixo, onde João é um individuo recuperado da OWL e é inferido que ele possui um irmão e não uma irmã.

Pessoa (João) ∧ temIrmãoOuIrmã (João,?x2) ∧ Homem (?x2) → temIrmão (João,?x2)

Também é possível aplicar as regras da SWRL utilizando valores literais, como podemos ver abaixo, onde é inferido que o irmão do João tem 40 anos de idade e é homem com base nos antecedentes.

Pessoa (João) ∧ temIrmãoOuIrmã (João,?x2) ∧ Homem (?x2) ∧ temIdade (?x2,40) → temIrmãoCom40Anos (João,?x2)

Também é possível através do uso do átomo *sameAs* inferir que dois indivíduos da OWL são o mesmo, como visto no exemplo abaixo:

sameAs (João, João Victor)

Da mesma forma o átomo *differentFrom* pode ser usado para inferir que os dois indivíduos não são o mesmo.

Além de permitir essa expressividade utilizando átomos definidos na própria OWL ou no SWRL, é possível utilizar alguns elementos incorporados (*built-Ins*) que são disponibilizados com as versões do SWRL. Esses *built-Ins* são baseados no reuso de outros existentes no *XQuery* e no *XPath*, que são ambos baseados no XML Schema e utilizam um *namespace* disponível através da W3C (HORROCKS et al., 2004).

Os *built-Ins* podem ser utilizados para diversas finalidades, entre elas podemos destacar (HORROCKS; PATEL-SCHNEIDER, 2003):

- comparação entre argumentos;
- cálculos matemáticos;
- valores booleanos;
- trabalhar com Strings;
- trabalhar com data, tempo e duração;
- trabalhar com URIs;
- e manuseio de listas.

2.4 Engenharia de Componentes

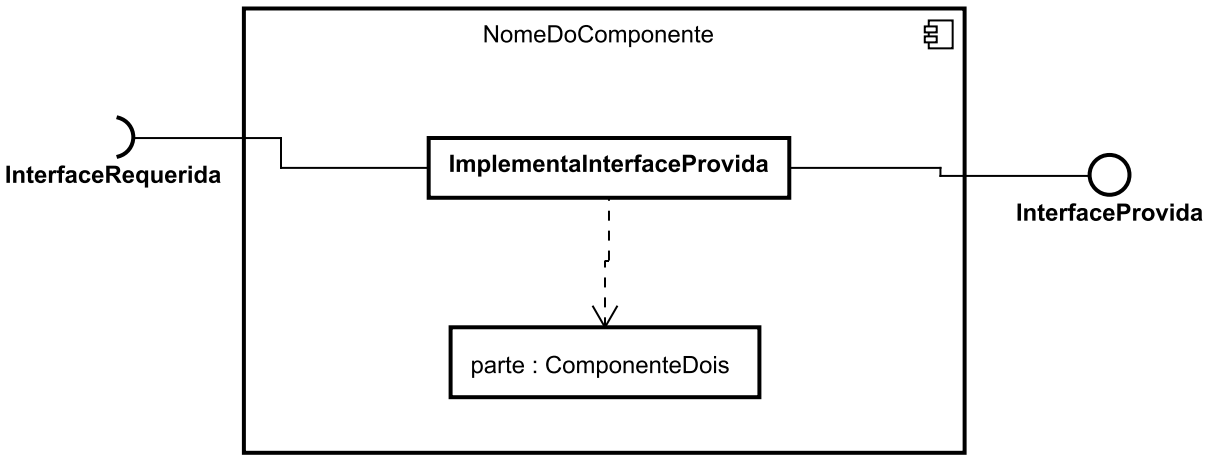
Segundo Braude (2005), um componente pode ser definido como um software utilizável sem alteração. No caso, um componente é um software que não pode ser alterado, porém suas instâncias podem. Um componente ainda pode ser definido como uma parte física substituível que de um sistema que empacota a implementação e fornece a realização de um conjunto de interfaces (OMG Object Management Group, 2010).

Uma parte física de um sistema pode ser representada como um componente, desde que ela possa ser substituída por outra parte sem que suas funções sejam modificadas. As interfaces disponibilizadas pelo componente devem ser capazes de garantir essa não modificação, pois tanto o componente a ser substituído quanto o que irá substituí-lo devem fornecer a mesma interface, permitindo assim manter seu funcionamento(BRAUDE, 2005).

2.4.1 Notação UML para Componentes

Através da notação UML para componentes é possível representar um componente através de um diagrama. Na Figura 6 é possível visualizar o exemplo de um diagrama de componentes e os principais elementos que o compõe.

Figura 6: Principais elementos do Diagrama de Componentes



Nessa figura podemos perceber que um componente pode prover e requerer interfaces, e ele pode possuir subcomponentes, que provém suas próprias interfaces. O sistema a utilizar o componente irá acessar as funcionalidades do componente através dos métodos da interface provida (BRAUDE, 2005).

2.4.2 Ciclo de vida de um Componente

O desenvolvimento de um componente ocorre de forma semelhante ao desenvolvimento de qualquer outro software. Primeiramente é necessário decidir quais partes do sistema a ser implementado devem ser projetadas em forma de um componente. Para isso é realizada a análise de domínio, que seria muito semelhante fase de análise do modelo clássico de desenvolvimento de software, porém nesse caso é preciso verificar quais partes do software da nossa aplicação precisamos e se essas partes podem ser utilizadas em aplicações semelhantes. Para isso é feito um levantamento das funcionalidades da aplicação e então feita essa verificação com base nessas funcionalidades (BRAUDE, 2005).

2.4.2.1 Projeto e Implementação

Um componente também é projetado de maneira semelhante a um software. Para projetá-lo é necessário criar o diagrama de componentes utilizando os elementos desse componente apresentados na Figura 6. Durante a criação desse diagrama são definidos os métodos que estarão presentes na interface a ser provida. Através desses métodos o

usuário do componente irá acessar as funcionalidades do módulo. Essa interface deve ser implementada no componente juntamente com as classes que ela utiliza para prover os serviços através dos métodos da interface.

Durante a implementação do componente deve-se assegurar que (BRAUDE, 2005):

- as interfaces e bibliotecas requeridas estão sendo importadas pelo componente;
- as regras do componente estão sendo atendidas;
- criar um documento que liste as partes do componente;

2.4.2.2 Criação de Instâncias

Após a criação do componente é feita a sua instanciação para verificar seus serviços providos. Para isso é criado uma classe de teste e é verificado se os retornos do módulo estão de acordo com o previsto através da interface que ele disponibiliza (BRAUDE, 2005).

2.4.2.3 Montagem

A montagem ocorre quando é necessário instanciar submódulos, criando novas funcionalidades através dos serviços providos pelos métodos das interfaces desses submódulos. Para prover essas novas funcionalidades deve-se criar uma interface para esse novo módulo com os novos métodos providos resultantes dessa instanciação (BRAUDE, 2005).

2.4.2.4 Instalação

Durante essa fase as interfaces providas pelo módulo são instanciadas pela aplicação que o utilizará, provendo assim os serviços do módulo. A instalação de um módulo pode ser realizada em diversas aplicações diferentes (BRAUDE, 2005).

2.4.2.5 Execução

Durante essa fase deve-se garantir a funcionalidade do módulo, garantindo assim que os serviços providos possam ser executados quando forem requeridos pela aplicação (BRAUDE, 2005).

2.5 API para Desenvolvimento

Para o desenvolvimento da camada de Semantica será utilizada a OWL API, e o pacote de consultas do *Apache Jena Jena.Query*.

Através da OWL API é possível representar as classes que representam a ontologia e seus elementos, permitindo assim sua manutenção (HORRIDGE; BECHHOFER, 2009).

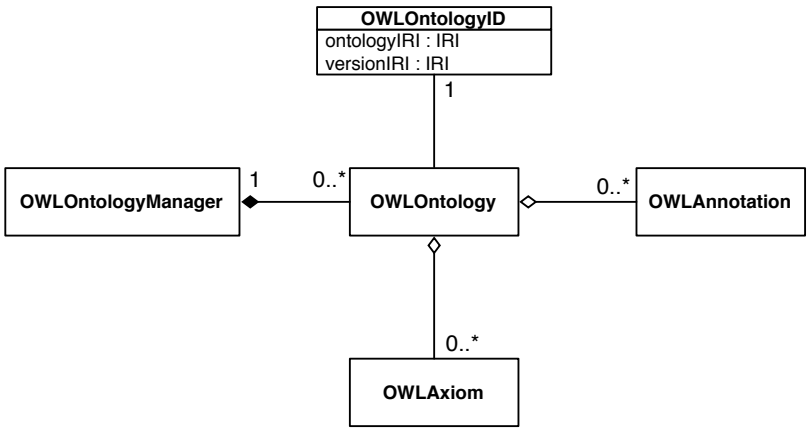
Através do processador de consultas *ARQ* é possível a execução de consultas SPARQL nos indivíduos presentes na ontologia.

2.5.1 OWL API

A OWL API é mantida pela Clark & Parsia LLC, uma empresa que fornece tecnologias para a Semanticca e o instituto de inteligencia artificial da Universidade de Ulm, instituição pública localizada no sul da Alemanha (OWL. . . ,).

A OWL API foi projetada para prover um conjunto de interfaces juntamente com implementações que facilitam o uso de ontologias. Através dessas interfaces é possível carregar as ontologias e salvar os indivíduos. Diferentemente das outras APIs como o Jena ou o Protégé 3.X API, a representação das expressões das classes e os axiomas não estão no formado de grafos RDF, ao invés disso ele é representado diretamente na especificação estrutural do OWL 2. Isso significa que a ontologia é vista como um conjunto de axiomas e anotações como visto na Figura 7

Figura 7: Modelo conceitual representando o gerenciamento da ontologia na OWL API



Fonte: Horridge e Bechhofer (2009, p. 3)

Enquanto as interfaces de modelo representam as entidades, as expressões das classes e os axiomas são representadas através de uma interface capaz de prover um ponto de acesso aos axiomas da ontologia. Existe também uma interface responsável somente pela criação, carregamento, mudança e salvamento das ontologias. Cada instância de uma ontologia possui seu gerenciador, e todas as mudanças a essa ontologia são aplicadas através desse gerenciador (HORRIDGE; BECHHOFER, 2009).

Esse sistema de gerenciamento centralizado permite que as aplicações tenham um ponto de acesso às ontologias, permitindo mecanismos de redirecionamento e outras

customizações para carregar as ontologias. Além disso, esse sistema centralizado permite que as aplicações cliente monitorem todas as mudanças que são aplicadas a qualquer uma das ontologia carregadas. O gerenciador também torna transparente ao usuário muitas das complexidades associadas com a escolha dos analisadores e renderizadores apropriados para a ler e salvar as ontologias (HORRIDGE; BECHHOFER, 2009).

A renderização e análise das ontologias da OWL 2 podem ser feitas através de várias sintaxes, não ficando preso ao RDF/XML. Pode ser usado por exemplo o RDF Turltle, que fornece uma serialização do RDF mais legível, ou a sintaxe do Manchester OWL, que disponibiliza uma serialização das ontologias que pode ser facilmente compreendida. Além disso o modelo da OWL API é bem subjacente, permitindo importar cláusulas de ontologias escritas em diferentes sintaxes (HORRIDGE; BECHHOFER, 2009).

Por padrão a OWL API disponibiliza estruturas de dados que permitem armazenar informações na memória em tempo de execução. Porém, para realizar as consultas de algumas informações essas devem ser armazenadas. Uma das formas de fazer o armazenamento dessas informações é através de mecanismos de armazenamento de triplas, onde triplas se refere ao fato das informações serem salvas no formado sujeito-predicado-objeto. A OWL API não possui esses mecanismos alternativos de armazenamento, porém ela permite que se utilize soluções de terceiros juntamente com a API (HORRIDGE; BECHHOFER, 2009).

Existem interfaces capazes de permitir a interação da API com raciocinadores. O raciocinador é responsável por verificar a consistência da ontologia. Essas interfaces foram projetadas para permitir que os raciocinadores possam verificar a ontologia sempre que ela for requisitada e salva, tornando a ontologia mais consistente. A interface principal para o raciocínio é a OWLReasoner, ela disponibiliza métodos para verificar a consistência das ontologias, computar a hierarquia e as propriedades das classes e verificar se os axiomas estão vinculados a ontologia (HORRIDGE; BECHHOFER, 2009).

Por fim, a OWL API possui uma API capaz de verificar a qual sublinguagem da OWL 2 a ontologia desenvolvida faz parte. Para isso existem algumas interfaces de sublinguagens que possuem funcionalidades que podem realizar essa verificação. Quando a verificação é feita, é reportado quais as clausuras incorretas na ontologia e em alguns casos o que está incorreto (HORRIDGE; BECHHOFER, 2009).

2.5.2 ARQ API - Processador de Consultas SPARQL

A ARQ API é mantida pelo *Apache Software Foundation*. Essa API foi projetada para prover interfaces que permitem a realização de consultas SPARQL (seguindo a recomendação da W3C) juntamente com classes que implementam essas interfaces. Desenvolvida para trabalhar como um motor de consultas para a JENA API. O principal

pacote dessa dessa API é o *jena.query*, neste pacote encontram-se as classes e interfaces responsáveis pela execução da consulta e seu retorno (Apache Software Foundation, 2013).

Segundo a Apache Software Foundation (2013), a ARQ API é capaz de realizar os quatro tipos principais de consultas SPARQL, que seriam:

- *SELECT:* retorna o iterador *ResultSet* do pacote *jena.query*, esse iterador é composto de objetos da interface *QuerySolution*, que podem ser acessados através dos métodos dessa classe;
- *ASK:* retorna um booleano correspondente a consulta realizada.
- *DESCRIBE:* retorna um objeto da interface *Model* do pacote *jena.rdf.model*, esse objeto contém um único grafo RDF contendo a resposta da consulta;
- *CONSTRUCT:* assim como o *DESCRIBE* retorna um objeto da interface *Model* do pacote *jena.rdf.model*, esse objeto contém um único grafo RDF contendo a resposta da consulta;

2.6 Fechamento do Capítulo

Foi apresentado neste capítulo alguns conceitos necessários para melhor entendimento do trabalho aqui apresentado. Através da execução do processo de inspeção de software é possível garantir a qualidade dos produtos de software. Porém a execução desse processo se torna difícil devido a sua sistemática.

Através do uso da Web Semântica é possível representar informações envolvidas em diferentes domínios. Essas informações são representadas através dos relacionamentos existente entre os conceitos envolvidos nesses domínios.

As ontologias nos permitem representar o conhecimento envolvido na Web Semântica. Através do uso de ontologias é possível processar informações de domínios computacionalmente.

O uso de componentes facilita a reusabilidade de sistemas, pois permite prover os mesmos serviços a diversas aplicações através das suas interfaces providas.

Através da OWL API e do Apache Jena ARQ é possível trabalhar com ontologias utilizando a linguagem de programação Java. A OWL API é responsável pela manutenção dos indivíduos de uma ontologia, e o Apache Jena ARQ nos permite realizar consultas SPARQL recuperando indivíduos de uma ontologia.

3 Trabalhos Relacionados

Neste capítulo é apresentado os trabalhos que possuem alguma relação com o trabalho proposto nesse documento. Na seção 3.1 é apresentada a metodologia de pesquisa utilizada para escolha dos trabalhos relacionados, também nessa seção esses trabalhos são apresentados e avaliados em duas categorias distintas. Por fim na seção 3.2 é feita uma breve conclusão sobre o que foi visto nesse capítulo.

3.1 Metodologia de Revisão Sistemática

Para pesquisar pelos trabalhos relacionados se utilizou uma metodologia de revisão sistemática, que consiste da delimitação do problema através de uma pergunta, neste caso, a pergunta definida foi *"quais soluções existem para dar suporte as atividades de inspeção de software de forma a aumentar a produtividade da empresa?"*. Após isso, definiu-se o *google scholar* como ferramenta de pesquisa em bases de dados. Definiu-se as seguintes palavras chaves para a pesquisa: *software, inspection, tool, optimization, quality, assurance, software engineering, improvement e ontology*. Para filtrar os resultados definiu-se os seguintes critérios: tamanho mínimo de 06 páginas, disponibilidade para download, apresentar resultados satisfatórios, ser alguma solução de engenharia de software, e por fim não ter passado mais de 08 anos da sua data de publicação.

Foram utilizadas somente palavras chaves em inglês pois não existem muitos trabalhos sobre o assunto em periódicos e revistas brasileiras, e acredita-se que periódicos internacionais possuem trabalhos mais bem conceituados que dão um maior valor para o trabalho aqui apresentado.

Após a pesquisa, encontraram-se 32 trabalhos relacionados, dos quais foram selecionados 07 que atendiam todos os critérios.

Os trabalhos relacionados encontrados foram divididos em duas categorias, na primeira categoria são apresentadas as soluções orientadas a automação do processo, que são soluções que de alguma forma automatizam uma fase ou etapa de algum processo de controle e garantia da qualidade. Na segunda categoria são apresentadas as soluções orientadas a melhoria de processo, onde o processo de inspeção é modificado para atender necessidades específicas, com o objetivo de aumentar a produtividade de tal processo.

3.1.1 Soluções Orientadas à Melhoria do Processo

Mishra e Mishra (2009), apresentam um processo de inspeção de software simplificado que ocorre de maneira semelhante a uma inspeção normal, tendo como principal

diferença o uso de uma ferramenta de inspeção durante a fase de preparação da inspeção. Essa ferramenta possibilita que o autor dos artefatos tenha acesso aos defeitos relatados pelos inspetores e possa comentá-los, diminuindo o esforço aplicado na reunião para decidir se um artefato está ou não com defeito. Para verificar a utilidade de tal processo foi feito uma comparação com o padrão IEEE e da acnasa (*National Aeronautics and Space Administration*), como resultado dessa comparação pode-se observar que o processo simplificado se equivale aos outros processos.

Lucia et al. (2011) propuseram uma adaptação ao processo de inspeção para uso em empresas distribuídas geograficamente. Para auxiliar nessa adaptação foi adotado o uso de duas ferramentas Web, a ferramenta de inspeção Web (WAIT), e o sistema avançado de gestão de artefatos (ADAMS), que foi integrado ao WAIT. Além disso, funcionalidades foram adicionadas ao ADAMS, permitindo que ele cubra todo o ciclo de vida do artefato. Para trabalhar em conjunto com essa ferramenta os autores definiram um processo de inspeção de software onde todas as fases da inspeção são registradas através do ADAMS. A ferramenta foi testada por alunos do curso de Ciencias da Computação da Universidade de Salermo, esses alunos relataram que a inspeção ocorreu de forma assíncrona, e que enquanto não fosse encontrado um numero minimo de defeitos os estudantes não realizavam a reunião de inspeção.

Mishra e Mishra (2012), nos mostram a dificuldade da aplicação de inspeções em empresas distribuídas geograficamente, segundo os autores uma dessas dificuldades é a realização das reuniões presenciais, pois elas não são viáveis nesse tipo de empresa. Então, para que essas empresas possam realizar a inspeção é proposto um processo de inspeção de software global que trabalha em conjunto com uma ferramenta Web desenvolvida pelos autores. Essa ferramenta permite que o líder da inspeção designe os inspetores ao projeto, e então eles devem reportar na ferramenta os defeitos encontrados nos artefatos, para que os autores possam comentar os defeitos, e, a reunião de inspeção é realizada virtualmente através da ferramenta.

3.1.2 Soluções orientadas à Automação do Processo

Lee e Wang (2009) propõe uma ferramenta multiagente baseada em ontologias de avaliação para o (*Capability Maturity Model Integration* (CMMI)). Para criar essa ferramenta o domínio do CMMI na área de garantia da qualidade do processo e do produto (*Process and Product Quality Assurance* (PPQA)) é representado em uma ontologia. Essa ontologia possui oito camadas, e três tipos de agentes. O primeiro agente é responsável pelo processamento de linguagem natural, ele rotula e filtra termos, o segundo, através do raciocínio ontológico captura os resultados do agente anterior, ele relaciona as forças dos termos com os conceitos do CMMI. O terceiro e ultimo agente é responsável pela sumarização, ele encontra conceitos, extrai-os, filtra os caminhos das sentenças, cria as

sentenças e calcula a taxa de compressão.

Wang e Lee (2008) propuseram neste trabalho um *Web service* inteligente para a área de garantia da qualidade do produto e do processo (PPQA) do CMMI. Esse Web service possui uma interface de usuário, um serviço de produtos de trabalho e processos de avaliação, um serviço de banco de dados, um agente de raciocínio ontológico, um prestador de serviço de visão objetiva, e um serviço de notificação de mensagens. Esse *Web service* tem como objetivo reportar através das interfaces de usuário os relatórios de não conformidades e de controle de qualidade, ele também mantém um registro das avaliações realizadas.

Nodler, Neukirchen e Grabowski (2009) apresentam o *XQuerybased Analysis Framework* (XAF), um *framework* em XML e XQuery para automatizar a análise de artefatos de software. Esse *framework* possui um alto nível de expressividade na inserção de regras de análise, ele é capaz de inspecionar vários artefatos de software sem a necessidade de adaptação das regras de análise. Como o XML é independente de formato ele é capaz de mapear os artefatos em forma de XML. Esse *framwork* é capaz de calcular medidas e detectar problemas em códigos fonte java, modelos UML e em casos de teste escritos utilizando *Testing and Control Notation - 3* (TTCN-3). Foi verificado a possibilidade da integração com uma IDE para torná-lo mais acessível. Através dessa integração foram realizados testes da ferramenta, e com base nesses testes a ferramenta mostrou-se satisfatória.

Silva et al. (2013) baseado na necessidade de automatizar as atividades da inspeção da garantia da qualidade apresentam um sistema baseado em ontologias e multiagentes chamado SystemQAI. Esse é capaz de automatizar a escolha de artefatos a serem inspecionados, o endereçamento das não conformidades, gerenciar cadastros e calcular indicadores de cobertura de inspeção, aderência ao processo e variação de esforço. Para verificar a efetividade proposta pela ferramenta foi realizado um experimento em uma fábrica de software, e através desse experimento com base nos indicadores foi possível verificar um aumento na cobertura dos artefatos a serem inspecionados, uma maior aderência ao processo de inspeção e uma leve diminuição no esforço aplicado para realizar a inspeção.

3.1.3 Análise dos Trabalhos Relacioados

A ferramenta proposta por (MISHRA; MISHRA, 2009) suporta as fases de inspeção privada e de controle, fases pertencentes ao processo proposto pelos autores. Em (MISHRA; MISHRA, 2012) os mesmos autores propuseram uma ferramenta que além de permitir a inspeção e controle torna a reunião de inspeção virtual, permitindo que a inspeção seja possível em empresas geograficamente distribuídas. Ambos processos e ferramentas conseguem cumprir seus objetivos, porém não permitem um controle maior dos artefatos, não dão suporte a criação de *checklists* e não permitem consulta de dados

estatísticos sobre as revisões, algo que a ferramenta proposta por (LUCIA et al., 2011) permite. Todas as ferramentas apresentam soluções que podem melhorar a produtividade da inspeção, porém o processo de inspeção ainda é realizado manualmente.

O XAF *framework* apresentado por (NODLER; NEUKIRCHEN; GRABOWSKI, 2009) é capaz de automatizar a análise dos artefatos de software na fase de inspeção, enquanto o SystemQAI, sistema proposto em (SILVA et al., 2013) automatiza a escolha de artefatos, o endereçamento das não conformidades, gerencia os cadastros e calcula indicadores. O XAF utiliza de XML e XQuery para automatizar suas tarefas, enquanto o SystemQAI utiliza OWL e multiagentes. Ambos são sistemas inteligentes, porém possuem diferentes focos. O XAF apesar de sua efetividade necessita que alguns artefatos a serem inspecionados sejam representados através de XML, isso limita seu poder. E o SystemQAI salva tanto os indivíduos da ontologia quanto os dados da inspeção na base de conhecimento, diminuindo assim a efetividade das consultas quando a base está bem populada. (WANG; LEE, 2008) e (LEE; WANG, 2009) apresentam trabalhos relacionados a avaliação do processo de inspeção. Ambos utilizam de agentes e ontologias para determinado fim. Seus processos são automatizados, e reportam aos usuários possíveis falhas e problemas em seus processos de inspeção. Apesar da produtividade da inspeção não ser o foco dessas ferramentas elas acabam contribuindo, pois ao saber das inconformidades do processo de inspeção é possível atuar em cima delas.

Dos trabalhos apresentados, os trabalho apresentado por Lee e Wang (2009), Wang e Lee (2008) e Silva et al. (2013) utilizam ontologias para representar o conhecimento envolvido durante os processos abordados. Ambos para trabalhar com ontologias utilizam diversos *frameworks* e APIs em conjunto com o sistema de inspeção. Dependendo de como essa implementação é realizada o sistema pode acabar ficando preso a tecnologia utilizada, dificultando assim a mudança de APIs e *frameworks*. Ao trabalhar com o módulo de Web Semântica aqui apresentado, que foi projetado e construído como um componente é garantido seu funcionamento caso seja necessária a utilização a troca do módulo por um que utilize tecnologias diferentes para chegar ao mesmo resultado, desde que nessa mudança seja assegurado que as interfaces providas pelo novo módulo sejam as mesmas do módulo substituído, mantendo assim o sistema funcional.

3.2 Fechamento do Capítulo

Neste capítulo foram apresentados os trabalhos relacionados ao trabalho aqui apresentado. Esses trabalhos abordam a automação de processos por parte de sistemas que utilizam de tecnologias para representar o conhecimento envolvido nesses processos, entre esses trabalhos alguns se destacam por utilizar ontologias para a representação desse conhecimento.

Durante a análise desses trabalhos foi possível verificar que alguns deles utilizam ontologias para representar o conhecimento envolvido nesses processos, validando assim a necessidade de um módulo que trabalhe com Web Semântica e permite o fácil acesso as informações contidas em uma ontologia.

4 Modulo de Web Semântica - SWeb Module

Neste capítulo é apresentada a arquitetura e as principais funcionalidades do modulo de Web Semântica, responsável pelo raciocínio da ontologia utilizando o raciocinador de ontologias *PelletReasoner* e pela manutenção e recuperação de indivíduos. Na seção 4.1 é apresentada uma visão geral desse módulo, explicando sua interação com outras aplicações. Na seção 4.2 é apresentado os serviços providos pelo módulo de Web Semântica. Na seção 4.3 é apresentada a arquitetura desse sistema, como foi feita sua divisão e como essas decisões foram tomadas. Por fim na seção 4.4 é apresentado o fechamento desse capítulo, onde é feita uma breve análise sobre o trabalho apresentado.

4.1 Visão Geral

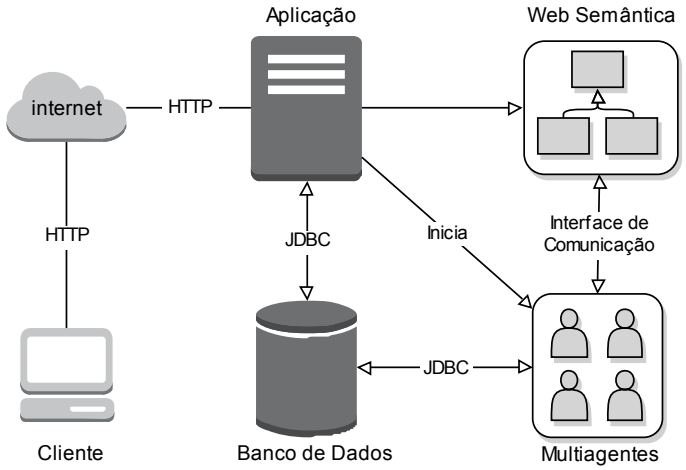
Conforme visto nos estudos apresentados neste trabalho, a inspeção de software é uma atividade que requer o envolvimento dos membros de uma equipe de desenvolvimento para ser realizada, utilizando assim parte do tempo desses membros, além disso ela é uma atividade sistemática. Devido a esses fatores, algumas empresas acabam por não realizar etapas da inspeção, diminuindo assim sua efetividade. Nos trabalhos relacionados apresentados anteriormente foram propostos sistemas para aumentar a produtividade de processos de garantia e controle de qualidade e do produto. Alguns desses trabalhos utilizam da Web Semântica para prover o conhecimento necessário para se realizar essas automações. Porém para prover a Web Semântica não é utilizado nenhum módulo ou componente, o acesso e manutenção dos indivíduos ocorre de forma direta, criando classes que trabalham somente com a ontologia a ser utilizada por esses projetos.

Sabendo-se do uso da Web Semântica para representar o conhecimento de processos, e a existência de poucas ferramentas que permitem o trabalho direto com a ontologia, sem a necessidade de trabalhar diretamente com as APIs e *frameworks* existentes, é apresentado neste trabalho a criação de um módulo de Web Semântica capaz de trabalhar com qualquer ontologia escrita na linguagem OWL (*Web Ontology Language*).

Esse módulo de Web Semântica faz parte de um projeto de pesquisa que tem por objetivo o desenvolvimento de um sistema de Suporte à Inspeção de Software. Na Figura 8 é possível visualizar a arquitetura desse sistema. Essa arquitetura é composta de quatro módulos, que seriam respectivamente a aplicação Web, o banco de dados relacional, um sistema multiagentes e um módulo de Web Semântica.

Dos módulos apresentados na Figura 8 a solução aqui apresentada é o módulo de Web Semântica. Nessa arquitetura podemos visualizar a interação do módulo de

Figura 8: Arquitetura do sistema de suporte à inspeção de software



Web Semântica com outras aplicações. Nela, o módulo se comunica com dois módulos diferentes do sistema, que seriam o sistema de multiagentes e a aplicação web. O acesso aos serviços do módulo pode ser feito através das interfaces de comunicação providas pelo módulo.

O módulo de Web Semântica é utilizado pela aplicação para poder popular a ontologia utilizada pelo Sistema de Suporte à inspeção. Para isso as interfaces que fornecem esse serviço são instanciadas na aplicação, e os métodos responsáveis pela população da ontologia são executados sempre que o usuário do sistema de suporte à Inspeção de Software achar necessário.

O Sistema Multiagentes utiliza o módulo de Web Semântica para consultar o conhecimento representado na ontologia, para isso consultas SPARQL são executadas através da interface interface provida pelo módulo.

4.2 Análise

Para projetar o módulo de Web Semântica primeiramente foi necessário definir os serviços oferecidos. Para isso foi feita uma análise dos serviços que podem ser providos por uma ontologia e então foram escolhidos os principais serviços providos pelo módulo e foi criada a seguinte lista de serviços a serem oferecidos:

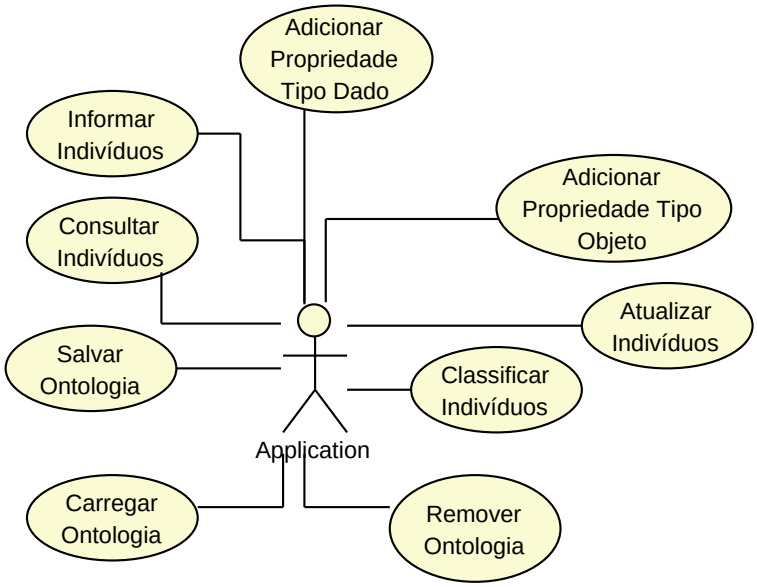
- *Prover acesso a ontologias:* permitir o acesso a ontologias existentes tanto na web quanto salvas localmente.
- *Prover acesso a consultas:* permitir que sejam realizadas consultas nessas ontologias, recuperando indivíduos, propriedades e informações.

- *Prover a manutenção dos indivíduos:* permitir que novos indivíduos sejam adicionados, removidos e atualizados na ontologia, juntamente com suas propriedades.

- *Prover mecanismos de inferência:* executar as regras de inferência existentes na ontologia e permitir acesso a essas regras.

Com base nessa lista se criou um diagrama de caso de uso, onde a interação das aplicações que utilizarem o sistema e as principais funcionalidades do módulo são representadas. Tal diagrama pode ser visto na Figura 9

Figura 9: Diagrama de Caso de Uso



Através desse diagrama de caso de uso é possível visualizar as principais funcionalidades disponibilizadas pelo módulo de Web Semântica. Os casos de uso apresentados neste diagrama representam as funcionalidades necessárias para prover os serviços do módulo de Web Semântica. Em seguida esses casos de uso são apresentados mais detalhadamente.

- *Carregar Ontologia:* fazer a leitura da ontologia através da internet ou de arquivos OWL mantidos em disco. As ontologias devem ser carregadas pelo módulo para que seja possível fornecer os serviços do módulo de Web Semântica.

- *Salvar Ontologia:* salvar em disco as modificações aplicadas na ontologia. A ontologia deve ser salva para que as modificações realizadas pelo módulo sejam persistidas. Caso a ontologia não seja salva, ao carregá-la novamente as alterações realizadas previamente não estarão disponíveis.

- *Remover Ontologia:* remover a ontologia do módulo, impedindo que ela seja utilizada, porém não excluindo-a do disco. Caso seja necessário a ontologia deve ser removida do módulo, evitando que alguma modificação seja persistida.

- *Informar Indivíduo:* informar um novo indivíduo a ontologia sem fazer sua persistência em disco. Os indivíduos devem ser informados a ontologia sem fazer suas persistências. Caso seja necessário a persistência dos indivíduos a ontologia deverá ser salva.

- *Classificar Indivíduo:* associar um indivíduo da ontologia à sua classe. Caso um indivíduo não seja associado a uma classe específica ele será um indivíduo da classe principal da ontologia (*Thing*), então para manter os indivíduos da ontologia organizados e evitar inconsistência nos dados da ontologia os indivíduos devem ser associados as suas respectivas classes.

- *Atualizar Indivíduo:* atualizar alguma informação do indivíduo. Os indivíduos da ontologia podem ser modificados a qualquer momento, reclassificando-os, alterando as propriedades deles ou adicionando novas propriedades.

- *Adicionar Propriedade Tipo Dado:* adicionar uma propriedade do tipo dado a um indivíduo. Uma das coisas que diferencia um indivíduo dos demais em uma ontologia são suas propriedades e relacionamentos. As propriedades do tipo dado do indivíduo devem ser associadas a ele, diferenciando assim ele dos outros indivíduos.

- *Adicionar Propriedade Tipo Objeto:* adicionar uma propriedade do tipo objeto a um individuo. Os indivíduos de uma ontologia se relacionam, esse relacionamento se dá através de suas propriedades do tipo objeto. Sendo assim, ao adicionar uma propriedade tipo objeto a um indivíduo ele está sendo relacionado a outro indivíduo.

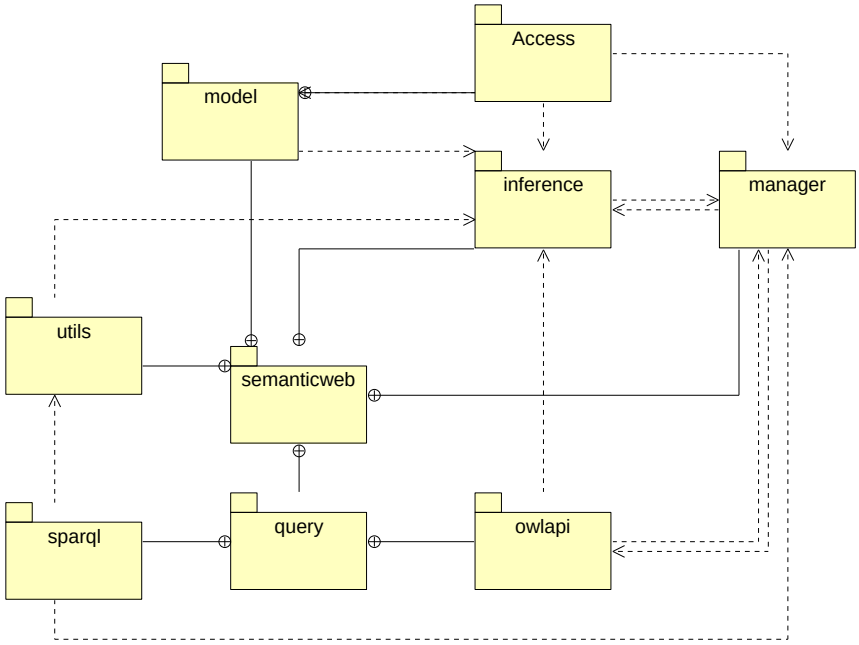
- *Consultar Indivíduos:* executar consultas nos indivíduos da ontologia retornando suas informações. Os indivíduos da ontologia devem ser recuperados através de consultas. Deve ser possível recuperar os indivíduos da ontologia tanto através de consultas SPARQL quanto através de consultas aos axiomas da ontologia.

4.3 Projeto do Sistema

O módulo de Web Semântica foi projetado e desenvolvido como um componente que pode ser utilizado por qualquer sistema. Devido as características de um componente o módulo de Web Semântica é independente do sistema que o utilizará. A arquitetura do modulo foi projetada através da engenharia de componentes. Esse módulo irá prover o conhecimento existente nas ontologias que nele forem carregadas.

Os serviços providos pelo módulo estão divididos em pacotes, onde as responsabilidades desses serviços são distribuídas para as classes e interfaces que os compõem. Na Figura 10 podemos ver essa organização de pacotes através de um Diagrama de Pacotes, onde é possível verificar a hierarquia existente entre os pacotes e a dependência entre as classes que os compõe. Abaixo é abordado as responsabilidades de cada um desses pacotes.

Figura 10: Diagrama de Pacotes



1. *semanticweb:* pacote principal, nele estão localizados todos os pacotes que fazem parte do módulo.

2. *manager:* contém as classes e interfaces responsáveis pela manutenção e leitura das ontologias.

3. *inference:* contém as classes responsáveis pela aplicação das regras de inferência na ontologia.

4. *query:* contém os os pacotes *sparql* e *owlapi*, responsáveis pela realização das consultas na ontologia.

5. *sparql:* contém a interface responsável pela execução das consultas SPARQL e sua implementação.

6. *owlapi:* contém a interface responsável pelas consultas realizadas através da *OWL API* e sua implementação.

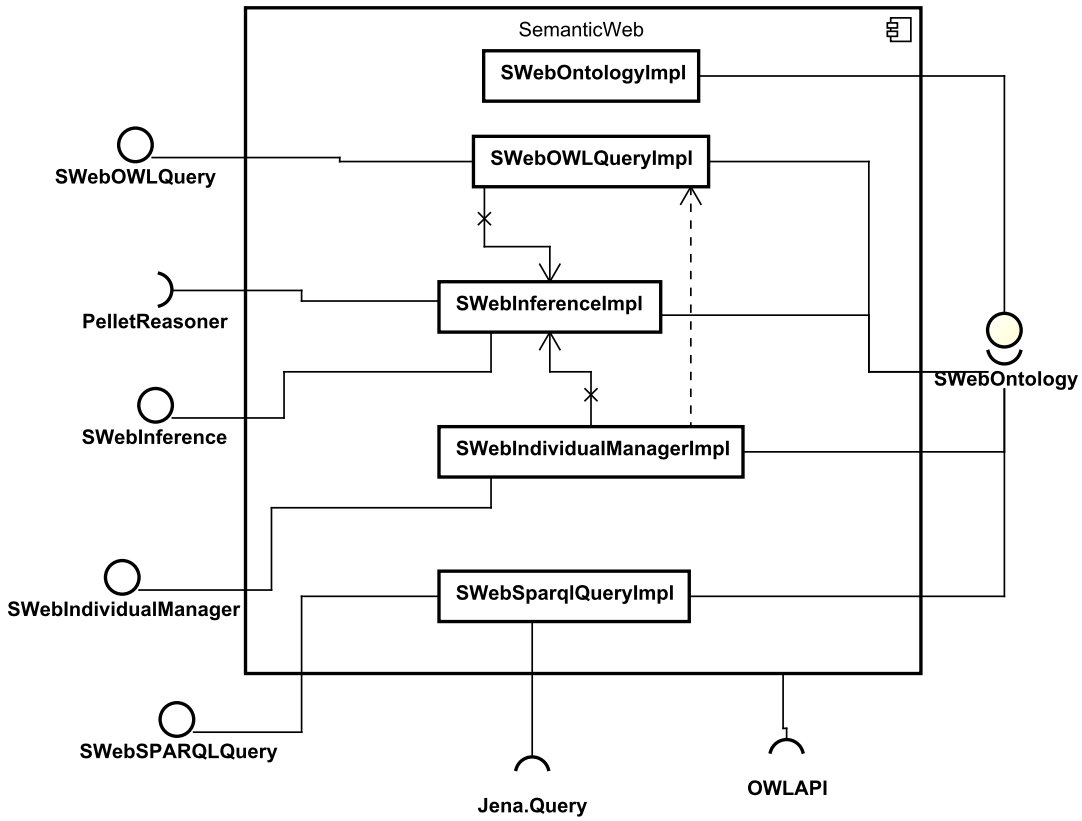
7. *model:* contém as classes responsáveis pela criação dos axiomas da ontologia e contém um pacote responsável pela persistência desses axiomas.

8. *access:* responsável pela persistência dos axiomas criados na classe *SWebIndividual*.

9. *utils:* contém classes de serviços utilizadas pelos outros pacotes.

Após a criação desse diagrama de pacotes, foram projetadas as interfaces fornecidas ao usuário para que ele tenha acesso ao módulo. Essas interfaces correspondem a vários *facades* que encapsulam os serviços disponibilizados pelas APIs e *frameworks* utilizados, assim como os serviços fornecidos pelas classes de serviço do módulo. Após a identificação e projeto dessas interfaces, criou-se o diagrama de componentes apresentado na Figura 11. Nele são apresentadas as interfaces requeridas, fornecidas e algumas classes que implementam essas interfaces.

Figura 11: Arquitetura do módulo de Web Semântica



A seguir é falado um pouco sobre cada uma dessas interfaces fornecidas e requeridas.

1. Interfaces Requeridas:

a) *OWL API:* utilizada por todo o módulo de Web Semântica, através dela é feita o gerenciamento das ontologias, a leitura dos indivíduos, a manutenção dos mesmos;

b) *PelletReasoner:* através do *PelletReasoner* é possível raciocinar as regras de inferência existentes na ontologia. Além disso ele é utilizado para adaptar a ontologia carregada pela OWL API para uma reconhecida pela *Jena.Query*, para que a mesma possa realizar consultas SPARQL;

c) *Jena.Query:* responsável pela realização das consultas SPARQL na ontologia, essas são fornecidas pelos usuários do módulo de Web Semântica através de arquivos de texto.

2. Interfaces Fornecidas:

a) *SWebOntology:* utilizada para fazer a leitura das ontologias, salvar as mesmas e remover uma ontologia do módulo, não permitindo assim que a mesma seja utilizada;

b) *SWebIndividualManager:* utilizada para manter os indivíduos da ontologia juntamente com suas propriedades;

c) *SWebSPARQLQuery:* utilizada para realizar consultas SPARQL nas ontologias carregadas a partir da interface SWebOntology;

d) *SWebOWLQuery:* utilizada para realizar algumas consultas simples a ontologia através da OWL API;

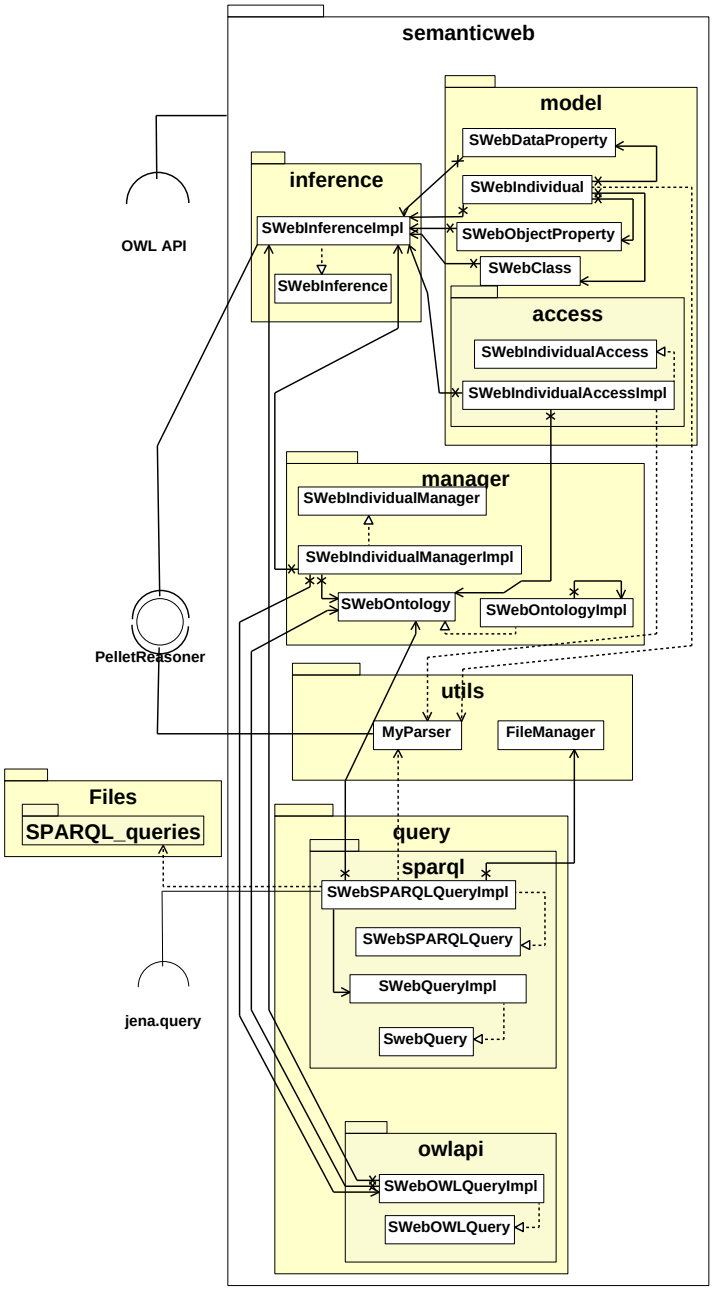
e) *SWebInference:* utilizada para listar as regras SWRL existentes na ontologia assim como listar os indivíduos que foram inferidos por essas regras.

Para representar as classes que serão utilizadas no sistema e seus relacionamentos foi criado um diagrama de classe onde os métodos e atributos estão ocultos para facilitar a sua compreensão. Esse diagrama de classe pode ser visualizado na Figura 12. No Apêndice A é possível visualizar esse diagrama de classe com todos seus atributos e métodos.

Neste diagrama de classe é possível visualizar as classes, interfaces, pacotes e relacionamentos que o módulo de Web Semântica possui. É apresentado também em forma de pacotes as API's (*Application Programming Interface*) externas utilizadas, e a pasta onde os arquivos contendo as consultas SPARQL são mantidos.

As classes apresentadas na Figura 12 são responsáveis por permitir que módulo de Web Semântica possa fornecer seus serviços. Em seguida é explicado como foi concebida a solução para cada um desses serviços.

Figura 12: Diagrama de Classe Simplificado



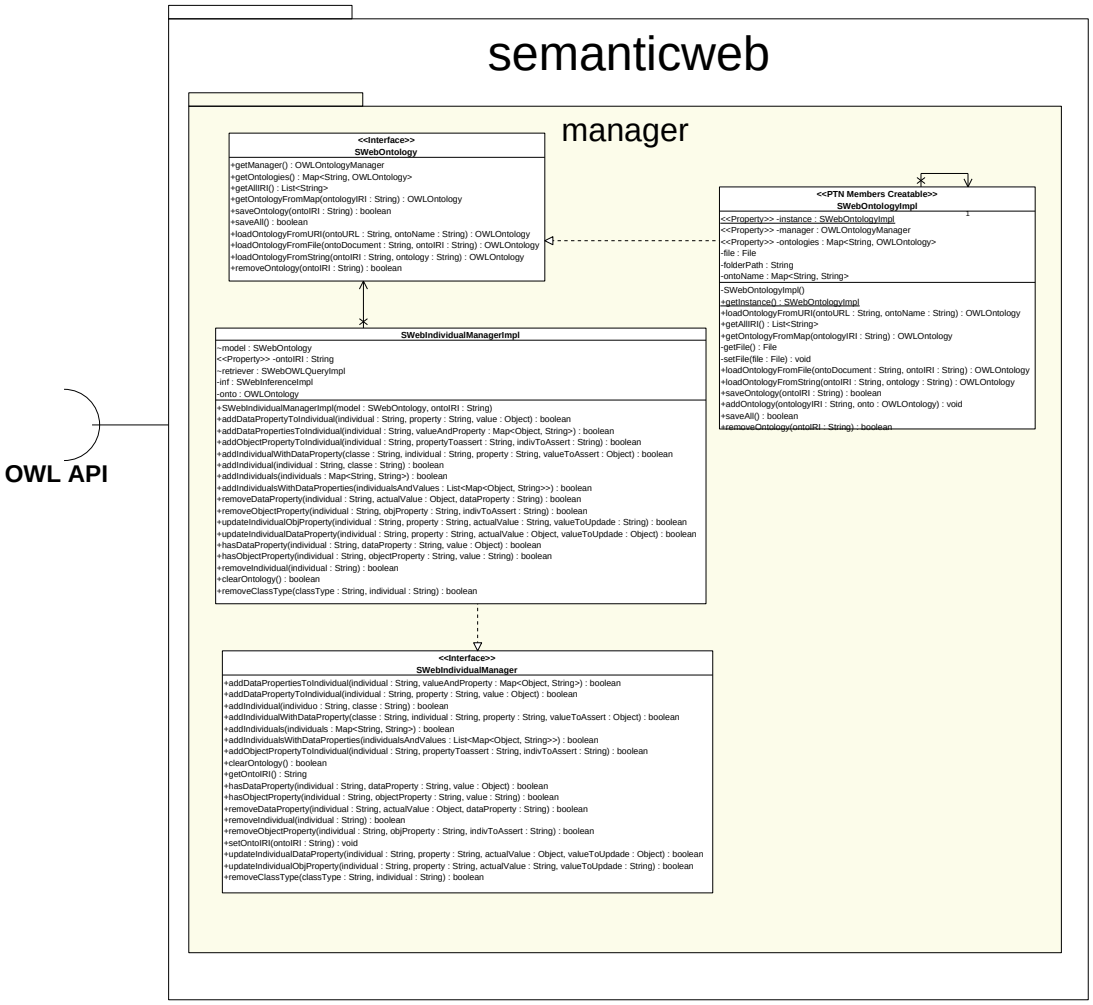
4.3.1 Implementação do Módulo de Web Semântica

O módulo de Web Semântica foi dividido por pacotes, onde cada pacote contém seus respectivos serviços. Para fornecer esses serviços foram criados *facades* que provém interfaces com os serviços do pacote. A seguir é apresentado como foi realizada a implementação desse *facades*.

4.3.1.1 Acesso as Ontologias

Para dar acesso as ontologias criou-se o pacote *manager*. Na Figura 13 é possível visualizar o diagrama de classe desse pacote.

Figura 13: Diagrama de Classe - Pacote *Manager*



Esse pacote contém entre as suas classes e interfaces a interface *SWebOntology*, e a classe *SWebOntologyImpl*, que são responsáveis pela leitura, persistência em disco e remoção das ontologias do módulo. Esse pacote foi criado para permitir que o usuário possa manter o controle das ontologias que estão sendo utilizadas.

A interface *SWebOntology* deve ser instanciada pelos usuários do módulo de Web Semântica. Para isso, foi utilizado o padrão de projeto *singleton* para o construtor da classe *SWebOntologyImpl* que implementa essa interface, permitindo assim que exista somente uma instancia dessa interface, evitando que as ontologias por ela carregadas sejam modificadas em diferentes instancias resultando na inconsistência dos dados.

A leitura das ontologias pode ser realizada de três formas diferentes, que seriam: através de arquivos *owl* salvos em disco no computador que utiliza o módulo, através da internet, e por último, através de campos de texto que contenham a ontologia.

Para realizar a leitura por meio de arquivos o usuário do módulo deve informar o local onde a ontologia está salva e a IRI da mesma, pois através dessa IRI é realizado o mapeamento das ontologias carregadas pela aplicação, esse processo pode ser visto no Apêndice B, onde através da classe *SWebOntologyImpl* é passado o local onde se encontra a interface e a IRI da mesma.

Para realizar a leitura de ontologias mantidas na internet o usuário deverá passar a URL onde essa ontologia se encontra e um local para salvar a mesma, pois caso o usuário queira salvar as modificações realizadas nessa ontologia será criado um arquivo com o caminho local.

Para a leitura de ontologias contidas em campos de texto é necessário informar ao sistema a IRI dessa ontologia e a mesma em um campo de texto.

Caso necessário, é possível remover as ontologias do módulo, não permitindo assim que alterações e consultas sejam realizadas sobre a população da mesma. Para isso é necessário informar a IRI da ontologia a ser removida por parâmetro para o método responsável pela remoção da ontologia chamado através de um objeto instanciado pela interface *SWebOntology*. Ainda caso necessário, o usuário do módulo pode salvar as alterações realizadas nas ontologias lidas, persistindo assim suas informações em disco.

4.3.1.2 Manutenção dos Indivíduos

A manutenção dos indivíduos também é realizada através do pacote *manager*, porém para isso foi fornecida a interface *SWebIndividualManager* juntamente com a classe *SWebIndividualManagerImpl* que implementa tal interface conforme pode ser visto na Figura 13.

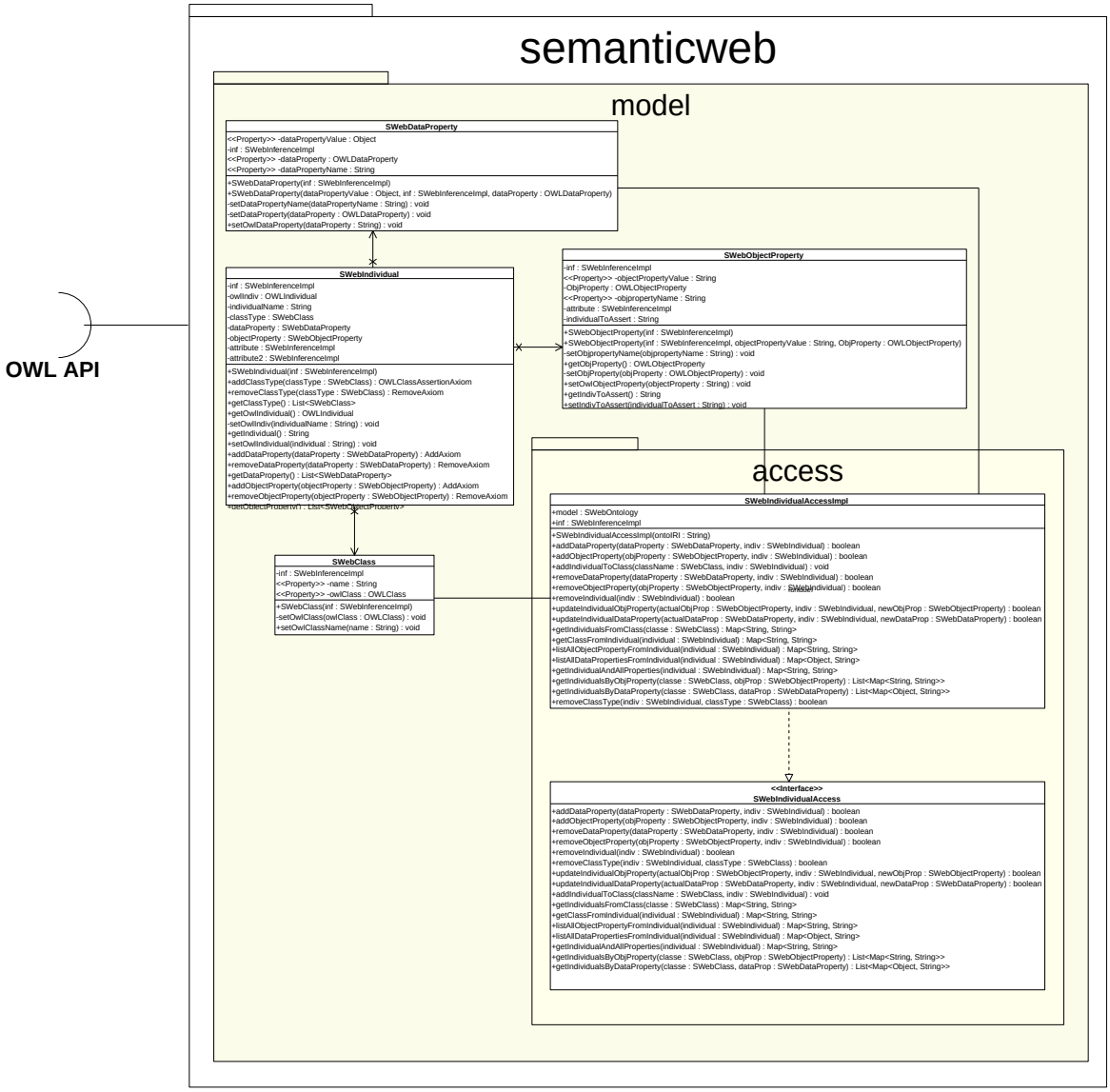
Através dos métodos da interface *SWebIndividualManager* é possível adicionar os indivíduos a ontologia, associar propriedades do tipo dado e do tipo objeto a esses indivíduos, definir a que classe esses indivíduos pertencem, remover essas definições de classes e de propriedades dos indivíduos e atualizá-los.

Para associar propriedades do tipo objeto aos seus indivíduos é necessário informar à interface *SWebIndividualManager* o nome do indivíduo a se adicionar a propriedades, o nome da propriedade, e o nome do indivíduo a ser associado através dessa propriedade.

Para realizar essas conversões e associações foram criados mais dois pacotes, o *model*, que possui as classes que representam os indivíduos, propriedades do tipo dado e objeto e as classes da ontologia e o pacote *access*, que possui a interface *SWebIndividualAccess* e a classe *SWebIndividualAccessImpl* que implementa tal interface. Na Figura 14

é possível visualizar as classes desses pacotes.

Figura 14: Diagrama de Classe - Pacotes *Model* e *Access*



As classes *SWebClass*, *SWebDataProperty*, *SWebObjectProperty* que compõem o pacote *model* têm por finalidade representar as propriedades e classes da ontologia. A classe *SWebIndividual* do mesmo pacote é responsável pela representação de um indivíduo da ontologia, essa classe possui métodos que criam os axiomas de inserção e remoção de propriedades e classes da ontologia.

Através da interface *SWebIndividualAccess* existente no pacote *access* é possível realizar a persistência desses indivíduos com seus axiomas na ontologia, salvando assim essas modificações em memória.

No Apêndice C é exibido um diagrama de sequência que exemplifica como um

indivíduo é adicionado a ontologia, para isso é necessário informar o nome desse indivíduo e o nome da classe que ele será associado, após isso é verificado se esses dados não estão vazios, e então é feita a criação dos objetos das classes *SWebClass* e *SWebIndividual*, onde são adicionados os valores a seus atributos, uma instância da interface *SWebIndividualAccess* é criada e esses dois objetos são passados por parâmetro a um método da classe *SWebIndividualAccess* que faz a persistência do indivíduo na ontologia carregada em memória. No Apêndice D e no Apêndice E são mostrados de forma semelhante como são adicionadas propriedades do tipo objeto e do tipo dado a esses indivíduos.

4.3.1.3 Recuperação de Informações da Ontologia

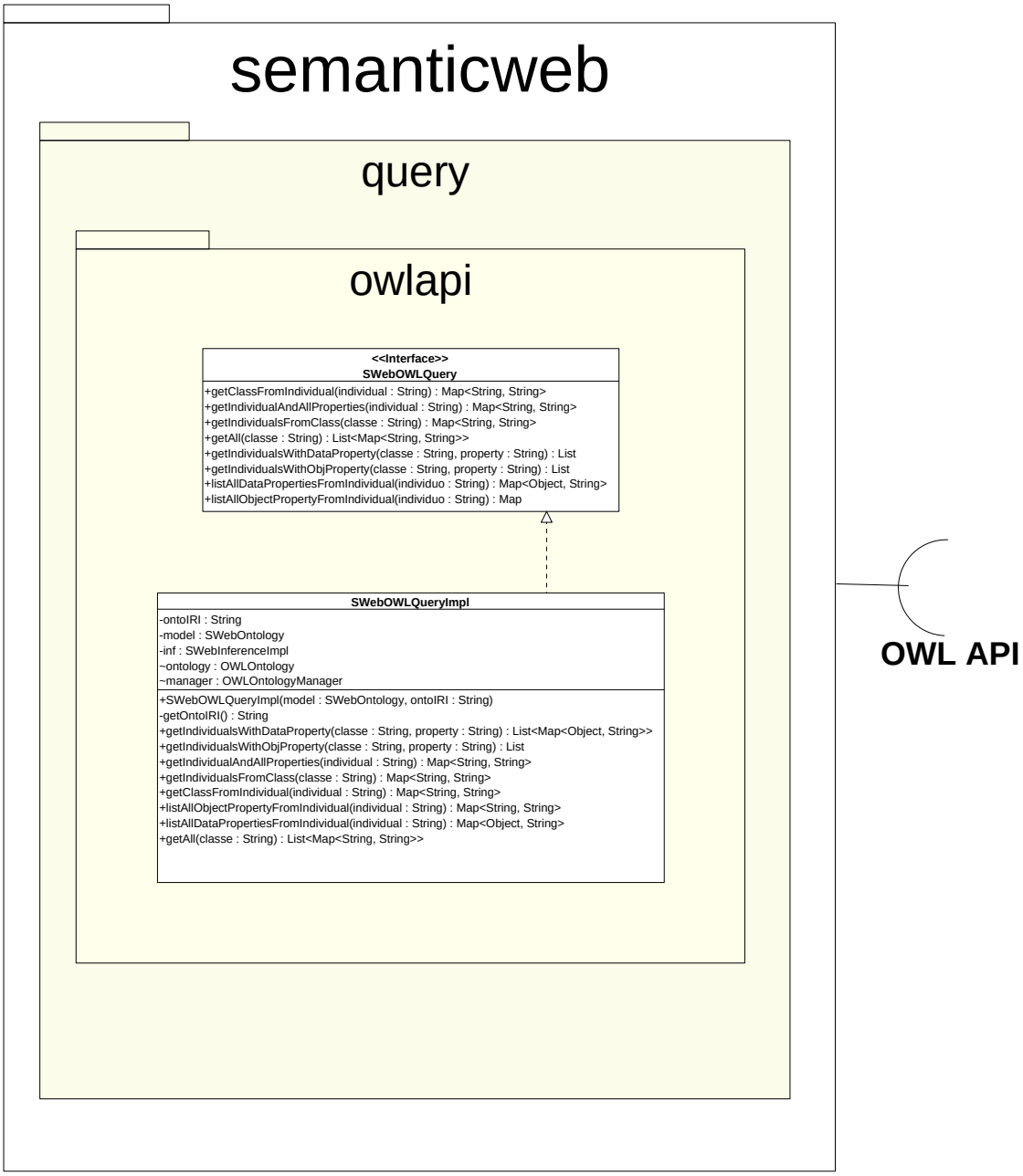
Para realizar a consulta dos indivíduos criou-se o pacote *query* e mais dois pacotes dentro dele, o *owlapi* e o *sparql*, que possuem diferentes meios para a recuperação de informações da ontologia. Através do pacote *owlapi* é possível realizar a recuperação somente de indivíduos e suas propriedades da ontologia. O pacote *sparql* permite a realização de consultas através da interface *jena.query*, que permite a realização de consultas SPARQL.

A recuperação dos indivíduos da ontologia realizado através do pacote *owlapi*, na Figura 15 é possível visualizar o diagrama com as classes desse pacote. As consultas pela OWL API ocorrem através da interface *SWebOWLQuery*, e da classe *SWebOWLQueryImpl*. Para isso são criados objetos das classes presentes no pacote *model* e passados para a interface *SWebIndividualAccess* através de métodos presentes na interface *SWebOWLQuery* . A interface *SWebIndividualAccess* processa esses objetos em seus métodos e retornar o que foi solicitado em mapa ou em uma lista de mapas, dependendo da solicitação realizada.

No Apêndice F é mostrado um diagrama de sequência que exemplifica melhor como essa recuperação de indivíduos e propriedades é realizada, nele é retornado uma lista de mapas que contém o indivíduo requisitado, as classes que ele pertence e as propriedades desse indivíduo. Através da chamada dos métodos existentes na interface SWebIndividualAccess é possível realizar a recuperação dos indivíduos da ontologia, e passar essas informações para a interface *SWebOWLQuery*, que retorna essas informações em uma coleção de dados ao usuário do módulo.

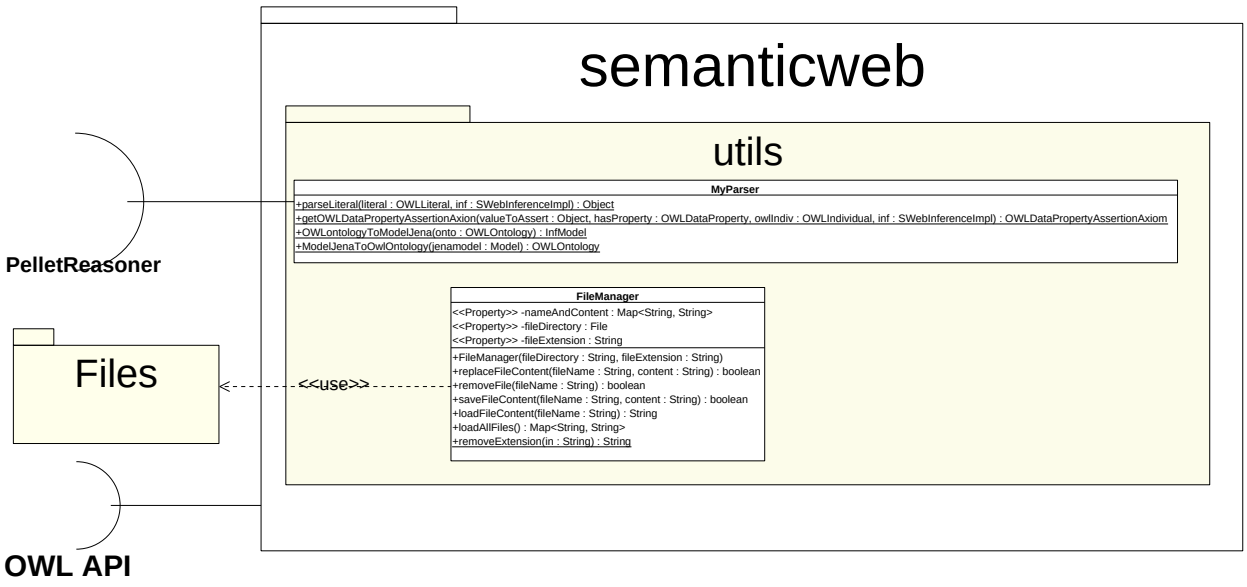
Para realizar as consultas SPARQL foi necessário criar um meio de converter os objetos que representam a ontologia lida através através da *OWL API* em objetos da interface *Apache Jena*, pois a OWL API não permite a realização de consultas SPARQL. Essa conversão é realizada através do *PelletReasoner* pela classe *MyParser*, permitindo assim a transformação de somente uma ontologia por vez e limitando a realização de consultas nas mesmas. Na Figura 16 é possível visualizar a essa classe responsável por esse serviço e a classe responsável pela manutenção das consultas SPARQL em arquivos de texto.

Figura 15: Diagrama de Classe - Classes do pacote *query.owlapi*



Através da interface *SWebSPARQLQuery* e da classe *SWebSPARQLQueryImpl* presentes no pacote *query.sparql* é possível executar consultas SPARQL nos indivíduos da ontologia. Na Figura 17 é possível visualizar o pacote *query.sparql* e as classes que o compõe. Para executar as consultas as mesmas devem ser armazenadas em arquivos dentro de uma pasta no sistema, e as mesmas devem seguir o padrão definido pela W3C para consultas SPARQL. Para realizar uma consultas o usuário do módulo deve instanciar a interface *SWebSPARQLQuery* através da classe *SWebSPARQLQueryImpl*, informando

Figura 16: Diagrama de Classe - Classes de Serviço do Pacote *utils*

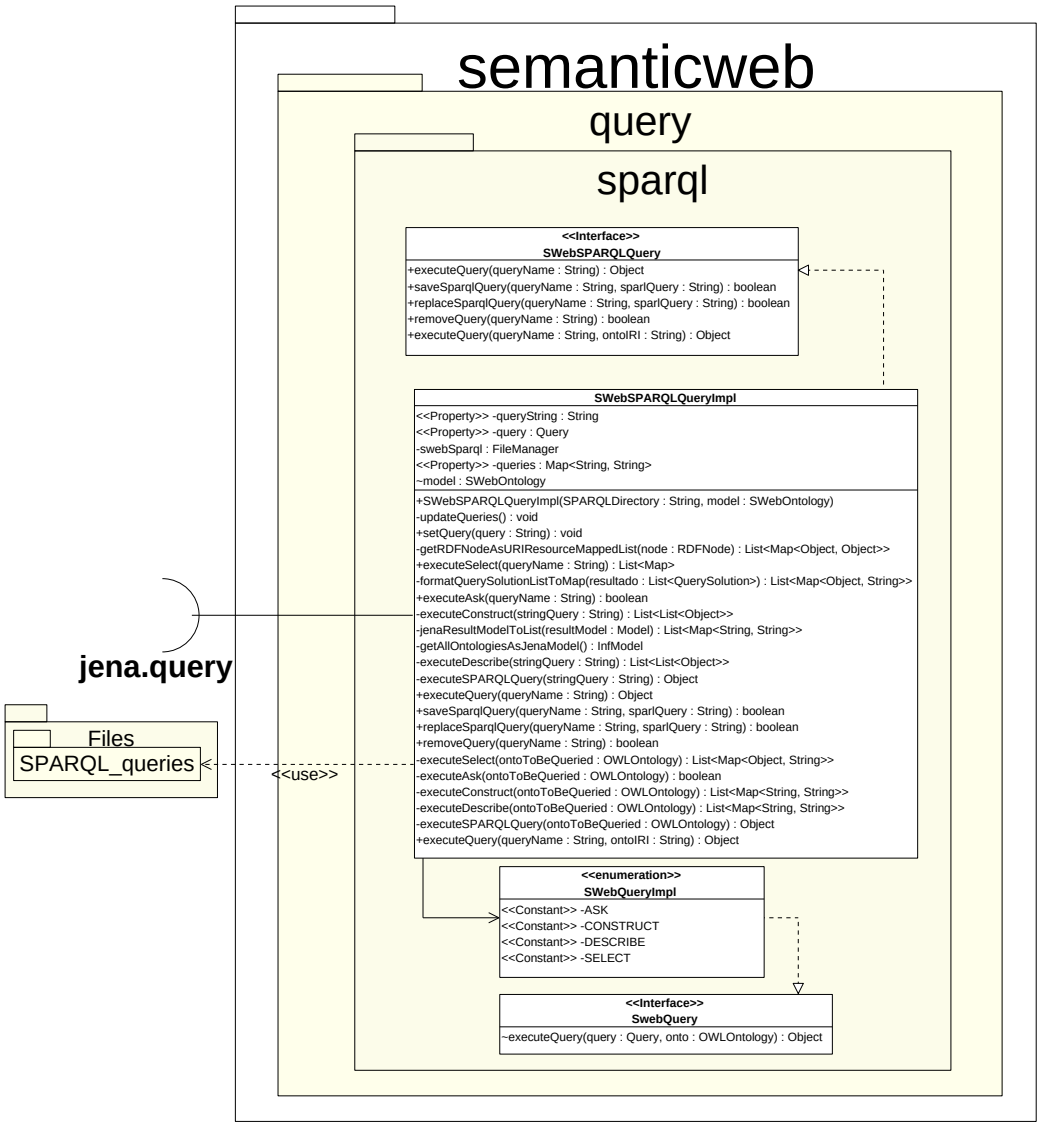


o diretório onde as consultas estão sendo mantidas e uma instância da interface *SWebOntology* para permitir a execução das consultas. Após isso, para executar as consultas nas ontologias o usuário deverá informar o nome da consultas e a IRI da ontologia que deseja consultar. Para realizar consultas dinâmicas onde apenas algumas variáveis da consulta são alteradas basta chamar o método da interface *SWebSPARQLQuery* responsável pela execução de consultas em campos de texto e passar a consulta em um campo de texto por parâmetro juntamente com a URI da ontologia a ser consultada.

Como existem quatro tipos de consultas que podem ser executadas, e para o usuário do módulo não é interessante informar qual tipo de consulta ele está realizando, já que essa informação fica explicita na consulta, criou-se utilizando o padrão *Strategy* uma estratégia de execução das consultas, onde através da interface *SWebQuery* é criado um método para execução de consulta que é implementado em cada um dos quatro elementos(*ASK*, *CONSTRUCT*, *DESCRIBE* e *SELECT*) do enumerador *SWebQueryImpl*, onde em cada elemento é realizado um tipo de consulta específico. Através da classe *SWebSPARQLQueryImpl* é verificado o tipo de consulta e então a interface *SWebQuery* é instanciada com o elemento respectivo da consulta, no Apêndice G é possível visualizar um diagrama de sequência onde esse processo é apresentado.

Através da interface *SWebSPARQLQuery* o usuário do módulo pode também salvar novas consultas SPARQL, substituir o conteúdo de algumas consultas já existentes ou removê-las fisicamente (do disco) e do sistema.

Figura 17: Diagrama de Classe - Classes Responsáveis Pelas Consultas SPARQL



4.3.1.4 Raciocínio das Ontologias

Através da interface *SWebInference* é possível realizar o raciocínio da ontologia e aplicar as regras de inferência nela existentes. A classe *SWebInferenceImpl* implementa tal interface. Através dessa interface o usuário da API pode listar as regras existentes na ontologia e os indivíduos que são afetados por essas regras. O raciocínio da ontologia é feito através da classe *SWebInferenceImpl* que instancia a interface requerida *PelletReasoner* responsável pelo raciocínio da ontologia quando o construtor da classe *SWebInferenceImpl* é chamado.

No Apêndice H é possível visualizar como o raciocínio da ontologia é realizado através de um diagrama de sequência, neste diagrama a classe que implementa a interface

SWebInference recupera a instância atual da interface *SWebOntology* através do método da classe *SWebOntologyImpl* que implementa o padrão *singleton*, e através dessa instância ele recupera a ontologia mapeada utilizando a IRI informada pelo usuário do módulo. Em seguida, através é chamado o método responsável pelo raciocínio da ontologia que verifica se a ontologia é consistente e então ele raciocina a mesma.

No Apêndice I é possível visualizar um diagrama de sequência que mostra como a listagem das regras é feita. No caso o módulo verifica através da OWL API quais os axiomas existentes na ontologia que está sendo raciocinada são representações de regras SWRL, e então ele retorna uma lista contendo esses axiomas convertidos em texto.

4.4 Fechamento do Capítulo

Neste capítulo foi apresentado o módulo de Web Semântica como solução para trabalhar com o gerenciamento de ontologias e seus indivíduos. Foi identificado através da engenharia de componentes a possibilidade de construir esse módulo como tal, então foram definidas interfaces para prover os serviços do módulo de Web Semântica e essas interfaces foram implementadas juntamente com outras classes que fornecem serviços a essas interfaces. Para a implementação desse módulo de Web Semântica foram requeridas interfaces capazes de gerenciar ontologias e realizar consultas.

5 Verificação e Validação do Módulo de Web Semântica

Neste capítulo são apresentados os processos utilizados para a verificação e validação do módulo de Web Semântica. Na seção 5.1 são apresentados os processos utilizados para a verificação e validação do módulo de Web Semântica, na seção 5.2 é apresentada a verificação do módulo de Web Semântica, na seção 5.3 é apresentado o processo executado para validar o módulo de Web Semântica. Por fim na seção 5.5 é apresentado o fechamento desse capítulo.

5.1 Processos De Verificação e Validação

Para verificar o módulo de Web Semântica foram executados testes unitários e testes de integração. Através desses testes é possível verificar se os métodos do módulo de Web Semântica estão funcionando corretamente, tanto em conjunto quanto individualmente.

Para realizar a validação do sistema foram realizados testes de sistema. Esses foram utilizados para verificar se o módulo está trabalhando em harmonia com o Sistema de Suporte à Inspeção de Software, verificando se os resultados gerados pelo mesmo são os esperados e se o módulo não ocasiona nenhuma falha no sistema de suporte.

Também foram utilizadas técnicas de revisões informais, onde durante o desenvolvimento do módulo de Web Semântica era verificado o código-fonte e as funcionalidades do módulo a procura de anomalias e falhas de serviços.

5.2 Verificação do Sistema

Os testes unitários e de integração foram criados utilizando a ferramenta de testes *TestNG*. Tal ferramenta foi escolhida por ser uma ferramenta de fácil entendimento e compreensão, além disso ela é uma ferramente de testes unitários e de integração completa.

Para a verificação da cobertura dos testes foi utilizado a ferramenta *Cobertura*, através dessa ferramenta é possível verificar quanto % das linhas de código estão sendo cobertas pelos testes, quais são essas linhas e quais *branchs* (estruturas condicionais, como *if/else* e *switch case*) estão sendo cobertos, permitindo assim um aumento na cobertura do código fonte.

Para fins de estudo dos resultados dos testes, dividiu-se as classes do módulo em

dois grupos distintos, o primeiro corresponde as que implementam as interfaces de serviços providas aos usuários do módulo, e o segundo seriam as classes que são utilizadas por essas implementações. Para facilitar a identificação desses dois grupos o primeiro grupo será chamado de *classes funcionais*, e o segundo de *classes de serviço*.

Foram realizados testes tanto para verificar se os métodos estão retornando o valor desejado a partir de uma requisição válida, quanto para verificar se o retorno é inválido quando a requisição é inválida. A partir desses testes foi possível verificar a cobertura dos testes, e fazer uma melhora constante nos testes a fim de tentar cobrir o máximo possível do código fonte do sistema.

Foram executados duas baterias de testes, onde em cada uma dessas baterias são testadas as classes do módulo e os métodos dessas classes. Para garantir essa verificação é foi feita a comparação do retorno de cada método (que possua retorno) com um resultado esperado.

5.2.1 Resultados dos Testes

Na primeira bateria foram executados 152 testes. Destes 152 testes, 60 são testes unitários, e 92 são testes de integração. Esses dados podem ser visualizados na Tabela 1.

Tabela 1: Primeira Bateria - Testes Unitários e de Integração Divididos por Tipos de Classes

Tipos de Teste	**Classes funcionais**	**Classes de Serviço**	**Total**
Testes Unitários	35	25	60
Testes de Integração	62	30	92
Total	97	55	**152**

Dos 60 testes unitários, 25 são testes aplicados em métodos das *classes de serviço*, e 35 são aplicados em *classes funcionais*.

Dos 92 testes de integração, 62 são aplicados nas *classes funcionais* e 30 em *classes de serviço*. Essa diferença ocorre pois a maioria dos métodos das *classes funcionais* utilizam os métodos das *classes de serviço* para fornecer seus serviços, e assim esses testes acabam se classificando como testes de integração.

Dos 152 testes unitários realizados 100% foram aprovados, conforme pode ser visto na Tabela 2. A taxa de 100% de aprovação dos testes se deve pois durante sua criação eles eram executados e os problemas encontrados eram corrigidos.

Após a execução dos testes verificou-se na ferramenta de cobertura que 82 % do código fonte e e 62 % dos *branchs* estavam sendo coberto pelos testes. Após a verificação do resultado das linhas de código fonte cobertas pelo teste verificou-se a existência de

Tabela 2: Primeira Bateria - Resultado dos Testes

	Testes Unitários	Testes de Integração
Aprovados	60	92
Reprovados	0	0

estruturas condicionais e blocos de *try* e *catch* que não estavam sendo cobertos. A não cobertura dessas estruturas acabou reduzindo a cobertura total do código fonte por parte dos testes. Na Figura 18 é apresentada uma visão geral dessa cobertura.

Figura 18: Primeira Bateria - Cobertura Geral dos Testes

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	25	82% 671/809	62% 139/224	1,831
br.edu.unipampa.issi.semanticweb.inference	3	94% 47/50	66% 4/6	1,105
br.edu.unipampa.issi.semanticweb.manager	4	83% 201/241	56% 60/106	2,517
br.edu.unipampa.issi.semanticweb.model	4	82% 87/106	50% 6/12	1,297
br.edu.unipampa.issi.semanticweb.model.access	2	98% 121/123	87% 28/32	1,545
br.edu.unipampa.issi.semanticweb.query.owlapi	2	100% 50/50	100% 2/2	1,056
br.edu.unipampa.issi.semanticweb.query.sparql	8	73% 77/105	44% 8/18	1,52
br.edu.unipampa.issi.semanticweb.utils	2	65% 88/134	64% 31/48	3,294

Em seguida os testes foram refatorados e novos testes foram construídos com o objetivo de aumentar a cobertura dos testes, para isso foram executados 187 testes em uma segunda bateria de testes. Desses 187 testes 65 são unitários e 122 são de integração.

Na Tabela 3 são apresentados os dados gerados por essa segunda bateria de testes.

Tabela 3: Segunda Bateria - Testes Unitários e de Integração Divididos por Tipos de Classes

Tipos de Teste	Classes funcionais	Classes de Serviço	Total
Testes Unitários	35	30	65
Testes de Integração	70	52	122
Total	105	82	**187**

Dos 187 testes executados 100% deles foram aprovados conforme pode ser visto na Tabela 4

Tabela 4: Segunda Bateria - Resultado dos Testes

	Testes Unitários	Testes de Integração
Aprovados	65	122
Reprovados	0	0

Com a execução dessa segunda bateria de testes ocorreu um aumento na quantidade de *branchs* e linhas de código fonte cobertos pela aplicação. Chegando a um resultado de 89% do código fonte coberto e 68% dos *branchs* conforme pode ser visto na Figura 19.

Figura 19: Segunda Bateria - Cobertura Geral dos Testes

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	25	89%	733/815	68%	157/230	1,842
br.edu.unipampa.issi.semanticweb.inference	3	94%	47/50	66%	4/6	1,105
br.edu.unipampa.issi.semanticweb.manager	4	86%	204/237	54%	59/108	2,534
br.edu.unipampa.issi.semanticweb.model	4	97%	102/105	75%	9/12	1,297
br.edu.unipampa.issi.semanticweb.model.access	2	100%	123/123	93%	30/32	1,545
br.edu.unipampa.issi.semanticweb.query.owlapi	2	100%	50/50	100%	2/2	1,056
br.edu.unipampa.issi.semanticweb.query.sparql	8	91%	106/116	77%	17/22	1,593
br.edu.unipampa.issi.semanticweb.utils	2	75%	101/134	75%	36/48	3,294

5.2.2 Análise dos Resultados

Após a execução das duas baterias de teste o número de *branchs* sendo coberto continuo baixo, inferior a 70%. Analisando os resultados da ferramenta *Cobertura* verificou-se que esse valor se deve a grande quantidade de estruturas condicionais e tratamento de exceções existentes nas classes do módulo, tornando difícil a cobertura das mesmas. Apesar do número baixo de *branchs* sendo cobertos foi possível cobrir cerca de 89% do código fonte da aplicação. A cobertura do código fonte é aceitável pois todos os métodos estão sendo cobertos, porém algumas condições deles não estão sendo atendidas.

5.3 Validação do Sistema

Para validar o módulo de Web Semântica instanciou-se o módulo através do Sistema de Suporte à Inspeção de Software, onde os serviços do providos pelo módulo foram validados através de testes empíricos, sendo observados os resultados obtidos pelo sistema através do módulo. Para isso primeiramente foi necessário popular a ontologia através do módulo de Web Semântica.

5.3.1 Resultados dos Testes

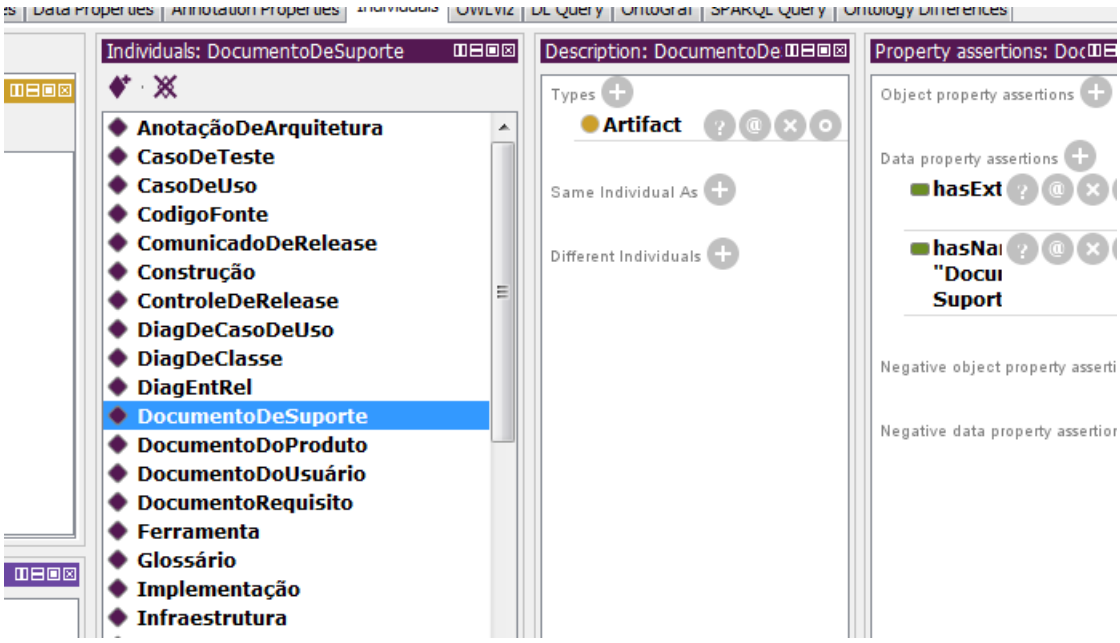
Para validar o módulo de Web Semântica primeiramente ele foi instanciado juntamente com o Sistema de Suporte à Inspeção de Software, no qual foram adicionados indivíduos e seus relacionamentos a ontologia carregada previamente pelo módulo.

Para verificar de uma forma externa a aplicação se a ontologia estava sendo corretamente populada foi utilizado o *Protégé* [1] Através dessa ferramenta foi possível visualizar

[1] *Protégé* é uma ferramenta desenvolvida para trabalhar com edição de ontologias, ela possui suporte

a população da ontologia após a ontologia ser populada e salva pelo Módulo de Web Semântica. Na Figura 20 é possível visualizar a população inserida na ontologia através do módulo de Web Semântica.

Figura 20: Novos Indivíduos da Ontologia Após População



Para realizar a recuperação dos indivíduos da ontologia o módulo de Web Semântica foi instanciado no módulo do sistema multiagentes, onde através da interface de consultas SPARQL foram realizadas as consultas aos indivíduos da ontologia.

Foram criadas duas consultas SPARQL, uma consulta contida num arquivo de texto salvo no disco do sistema, e outra salva em memória em um campo de texto.

Na Figura 21 é exibida a consulta salva em disco aplicada para recuperar os produtos de trabalho da ontologia. Através dessa consulta são recuperados os tipos de produtos de trabalho com seus nomes, sufixos, prefixos e extensão de arquivo. Essa consulta é utilizada pelo Sistema Multiagentes para identificar a entrega dos artefatos via *git*(Sistema de controle de versionamento distribuído) e sugerir os artefatos a serem inspecionados em determinada fase.

Na Figura 22 é possível visualizar o artefato localizado pelo sistema multiagentes ao executar a consulta SPARQL através do módulo de Web Semântica.

Na Figura 23 é exibida a consulta mantida em um campo de texto no módulo multiagentes, a mesma é realizada para recuperar as descrições dos itens de revisão da

completo à OWL 2 e através de raciocinadores como *HermiT* e *Pellet* ela mantém a ontologia em memória melhorando o desempenho do trabalho com a mesma (Stanford Center for Biomedical Informatics Research, 2014).

Figura 21: Consulta SPARQL que recupera os Produtos de Trabalho da Ontologia

```
PREFIX insp: <http://www.semanticweb.org/onto-inspection.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?prefix ?sufix ?extension
WHERE
        {?ind insp:hasName ?name.
        ?ind insp:hasExtension ?extension.
        OPTIONAL
        {?ind insp:hasPrefix ?prefix.
        ?ind insp:hasSufix ?sufix}
        }
```

Figura 22: Artefato Localizado Pelo Sistema Através do Resultado da Consulta SPARQL

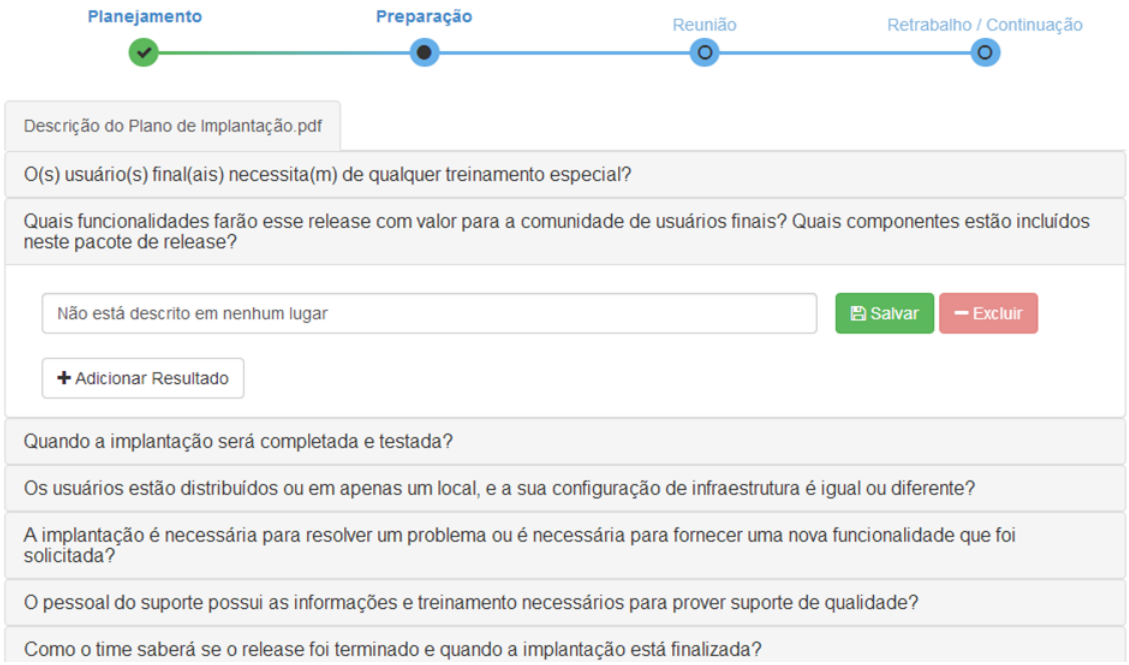
ontologia juntamente com o nome do artefato relacionado a ele. O campo entre colchetes *[workProduct]* é substituído durante a realização da consulta pelo nome do produto de trabalho no qual se deseja recuperar os itens de revisão. Através do resultado dessa consulta é possível identificar quais itens de revisão devem fazer parte do *checklist* a ser aplicado no artefato a ser inspecionado durante o processo de Inspeção de Software.

Figura 23: Consulta SPARQL que recupera os Intens de Revisão da Ontologia

```
PREFIX insp: <http://www.semanticweb.org/onto-inspection.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?prefix ?sufix ?extension
WHERE
        {?ind insp:hasName ?name.
        ?ind insp:hasExtension ?extension.
        OPTIONAL
        {?ind insp:hasPrefix ?prefix.
        ?ind insp:hasSufix ?sufix}
        }
```

Na Figura 24 é possível visualizar o *checklist* montado pelo sistema Multiagentes com base no resultado da consulta SPARQL executada pelo módulo de Web Semântica.

Figura 24: *Checklist* Montado Pelo Sistema Utilizando o Resultado da Consulta SPARQL



5.3.2 Análise dos Resultados

O módulo de Web Semântica forneceu todos os serviços requisitados pelo Sistema de Suporte à Inspeção de Software em um tempo quase instantâneo. Além disso durante sua execução não ocorreram falhas tanto no módulo quanto no sistema de suporte.

Durante a execução do Sistema de Suporte à Inspeção de Software não houve nenhum erro em relação a execução das consultas SPARQL na ontologia. Infelizmente o Sistema de Suporte à Inspeção de Software não utiliza por completo o módulo de Web Semântica, pois algumas funcionalidades como a remoção de indivíduos, consultas simples através da OWL API, o uso de *DESCRIBE*, *ASK* e *CONSTRUCT* em consultas SPARQL, entre outras não foram realizadas.

Assim é possível observar que o módulo de Web Semântica através de suas interfaces providas consegue fornecer os serviços necessários aos sistemas que o utilizar. Através do módulo de Web Semântica foi possível recuperar parte do conhecimento necessário para que o sistema de suporte à inspeção possa automatizar algumas atividades do processo de inspeção de software.

5.4 Revisões Informais

Foram realizadas revisões informais no código-fonte e nas funcionalidades do módulo com o objetivo de procurar inconsistências com padrões pré-estabelecidos e com os

serviços a serem providos.

Essas revisões informais foram aplicadas durante a fase de desenvolvimento do módulo, onde não era feito nenhum registro dessas revisões.

Para o código fonte do módulo foram feitas revisões informais procurando manter alguns padrões de codificação e manter a consistência do código com a arquitetura projetada, logo os seguintes padrões foram estabelecidos e revisados no código:

- idioma de codificação: todas as classes e métodos foram escritas em inglês com o objetivo de padronizar o módulo e se possível disponibilizá-lo conforme as regras vigentes na instituição;
- classes: todas as classes do módulo devem ter seu nome iniciado em letra maiúscula e caso seu nome seja composto ele deve ter a primeira letra de cada palavra maiúscula;
- padrões de projeto: os padrões de projetos especificados na arquitetura do sistema devem ser utilizados corretamente;
- comentários: os comentários do código devem ser claros e compreensíveis;
- atributos: os atributos do código devem ser iniciados em minúsculo e caso sejam compostos de mais de uma palavra as demais devem ser escritas com sua letra inicial em maiúsculo.

Para revisar as funcionalidades do módulo ele foi instanciado e os resultados de seus métodos foram verificados com base em resultados esperados, onde caso algum problema fosse detectado o mesmo era prontamente corrigido, mantendo assim o funcionamento do módulo. Além disso, também foi revisado com base na especificação do módulo se os seus serviços providos estavam de acordo com os requisitados pelo sistema de suporte à inspeção de software e se tais serviços eram genéricos suficientes para serem utilizados por qualquer outro sistema que necessitasse do acesso à Web Semântica.

5.5 Fechamento do Capítulo

Neste capítulo foram apresentadas as técnicas de validação e verificação utilizadas para validar o módulo de Web Semântica. Através do uso dessas técnicas foi possível identificar erros na implementação que foram corrigidos durante sua implementação.

A realização desses testes agregou um grande valor ao módulo, pois através deles é possível prever algumas possíveis falhas no sistema e garantir que tais falhas não ocorram durante a utilização do módulo.

Apesar da revisão do módulo ter ocorrido de forma informal e não ter compreendido todos os artefatos gerados pelo módulo ela foi capaz de manter alguns padrões na codificação do módulo e manteve a coerência dos serviços do módulo com o que havia sido projetado e requisitado.

6 Considerações Finais

O uso de ontologias para representar o conhecimento envolvido em processos vem se tornando cada vez mais constante. Através dessas representações é possível recuperar o conhecimento existente nesse processo, permitindo assim sistemas realizem atividades automaticamente sem a necessidade da interação humana. Porém para os sistemas computacionais poderem manusear uma ontologia é necessário que eles utilizem diversas ferramentas e APIs. Com o objetivo de realizar a integração dessas ferramentas em um único componente capaz de fornecer os serviços necessários para acessar as ontologias e seus indivíduos se apresentou neste trabalho um módulo de Web Semântica que fornece esses serviços através de suas interfaces providas.

A identificação dos serviços do módulo de Web Semântica foi feita através da análise dos serviços necessários ao Sistema de Suporte de Inspeção de Software, após essa identificação verificou-se a possibilidade da criação de um componente que possa ser substituível e utilizado por outros sistemas. A partir dessa identificação o módulo de Web Semântica foi projetado como um componente que provêm e solicita interfaces para realizar os seus serviços. Após o desenvolvimento desse módulo ele foi testado através de testes unitários e de integração até que seus erros estejam todos corrigidos. Após essa correção foi feita a instanciação do módulo no Sistema de Suporte à Inspeção de Software onde ele foi validado de forma empírica através da verificação dos seus serviços oferecidos.

Através do uso da *OWL API* em conjunto com o raciocinador *Pellet* e a interfaces de consulta *Jena Query* foi possível realizar a criação desse módulo de forma a permitir a execução de consultas e a manutenção dos indivíduos da ontologia carregada pelo módulo.

Apesar de fornecer os principais serviços para se trabalhar com a Web Semântica através do uso de ontologias o módulo aqui apresentado possui algumas limitações. Ele não permite a criação de regras SWRL, ele apenas as interpreta, limitando assim sua utilização, pois caso seja necessário a criação de regras SWRL dinâmicas por meio do módulo isso não é possível. O módulo também não é capaz de executar consultas SPARQL em mais de uma ontologia ao mesmo tempo, essa limitação ocorre pois a OWL API não permite a execução de consultas SPARQL, sendo necessário trabalhar com o JENA, e o método de conversão de objetos que representam a ontologia não permite tais consultas.

Os testes aplicados no módulo garantem suas funcionalidades, porém uma validação mais abrangente é necessária, pois o Sistema de Suporte à Inspeção de Software não utiliza o módulo por completo.

A criação do módulo de Web Semântica permitiu a utilização de padrões de projetos como o *Singleton*, o *Strategy* e o *Facade*. Através desses padrões foi possível prover

os serviços do módulo de Web Semântica. Através do trabalho aqui apresentado também foi possível trabalhar com o desenvolvimento de um software como um componente, oportunidade que não havia sido disponibilizada anteriormente.

6.1 Trabalhos Futuros

Após a realização desse trabalho percebeu-se que algumas mudanças e adições de serviços poderiam ser realizadas para melhorar o módulo de Web Semântica. Acredita-se que realizando essas modificações a utilização do módulo será mais eficaz. A seguir algumas dessas modificações são apresentadas.

É proposto para ser realizado em um trabalho futuro a criação de métodos e classes que permitam a execução de consultas SPARQL em mais de uma ontologia ao mesmo tempo. A OWL API não possui suporte nativo a consultas SPARQL, logo essa proposta é um desafio, pois será necessário criar meios para permitir que mais de uma ontologia seja consultada por vez.

A interface *SWebInference* atualmente oferece somente serviço de consultas as regras SWRL existentes no sistema e a indivíduos que são inferidos por essas regras. Logo, é também é proposto como um trabalho futuro a criação de métodos nessa interface que permitam a informação de novas regras SWRL a ontologia carregada.

Referências

ANTONIOU, G.; HARMELEN, F. V. Web ontology language: Owl. In: *Handbook on ontologies*. [S.l.]: Springer, 2004. p. 67–92. Citado na página 34.

Apache Software Foundation. *ARQ - A SPARQL Processor for Jena*. 2013. Disponível em: <https://jena.apache.org/documentation/query/>. Citado na página 44.

AURUM, A.; PETERSSON, H.; WOHLIN, C. State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 12, n. 3, p. 133–154, 2002. Citado 2 vezes nas páginas 23 e 27.

BARNERS-LEE, T. *Artificial Intelligence and the Semantic Web*. W3C Web Site, 2006. Visitado em 16/09/2013. Disponível em: <http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html>. Citado na página 33.

BERNERS-LEE, T. et al. The semantic web. *Scientific american*, New York, NY, USA:, v. 284, n. 5, p. 28–37, 2001. Citado 2 vezes nas páginas 31 e 32.

BRAUDE, E. *Projeto de software: da programação à arquitetura: uma abordagem baseada em Java*. [S.l.]: Bookman, 2005. Citado 3 vezes nas páginas 39, 40 e 41.

CHANDRASEKARAN, B. et al. What are ontologies, and why do we need them? *IEEE Intelligent systems*, v. 14, n. 1, p. 20–26, 1999. Citado na página 24.

DENGER, C.; SHULL, F. A practical approach for quality-driven inspections. *Software, IEEE*, IEEE, v. 24, n. 2, p. 79–86, 2007. Citado 2 vezes nas páginas 24 e 25.

FAGAN, M. E. Design and code inspections to reduce errors in program development. In: *Pioneers and Their Contributions to Software Engineering*. [S.l.]: IBM Systems Journal, 1999. v. 38, p. 258–287. Citado 2 vezes nas páginas 23 e 25.

FALBO, R. A. et al. Ontologias e ambientes de desenvolvimento de software semânticos. *Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, 2004. Citado na página 32.

FREITAS, F. L. G. de. Ontologias ea web semântica. *Sociedade Brasileira de Computação (SBC). II Jornada de Mini-Cursos de Inteligência Artificial*, p. 1–52, 2003. Citado na página 31.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, Elsevier, v. 43, n. 5, p. 907–928, 1995. Citado 2 vezes nas páginas 24 e 33.

HEDBERG, H.; LAPPALAINEN, J. A preliminary evaluation of software inspection tools, with the desmet method. In: IEEE. *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*. [S.l.], 2005. p. 45–52. Citado na página 24.

HORN, A. On sentences which are true of direct unions of algebras. *Journal of symbolic logic*, JSTOR, p. 14–21, 1951. Citado na página 38.

HORRIDGE, M.; BECHHOFER, S. The owl api: A java api for working with owl 2 ontologies. In: *OWLED*. [S.l.: s.n.], 2009. v. 529, p. 11–21. Citado 3 vezes nas páginas 41, 42 e 43.

HORROCKS, I.; PATEL-SCHNEIDER, P. F. Reducing owl entailment to description logic satisfiability. In: *The Semantic Web-ISWC 2003*. [S.l.]: Springer, 2003. p. 17–29. Citado 2 vezes nas páginas 36 e 39.

HORROCKS, I. et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, v. 21, p. 79, 2004. Citado 2 vezes nas páginas 38 e 39.

IEEE Standard for Software Reviews and Audits. *IEEE STD 1028-2008*, p. 1–52, 2008. Citado 6 vezes nas páginas 23, 27, 28, 29, 30 e 31.

KLYNE, G.; CARROLL, J. J.; MCBRIDE, B. Resource description framework (rdf): Concepts and abstract syntax. *W3C recommendation*, v. 10, 2004. Citado na página 34.

KOLLIA, I.; GLIMM, B.; HORROCKS, I. Sparql query answering over owl ontologies. In: *The Semantic Web: Research and Applications*. [S.l.]: Springer, 2011. p. 382–396. Citado na página 36.

LEE, C.-S.; WANG, M.-H. Ontology-based computational intelligent multi-agent and its application to cmmi assessment. *Applied Intelligence*, Springer US, v. 30, n. 3, p. 203–219, 2009. ISSN 0924-669X. Disponível em: <http://dx.doi.org/10.1007/s10489-007-0071-1>. Citado 2 vezes nas páginas 46 e 48.

LIBRELOTTO, G. R.; RAMALHO, J. C.; HENRIQUES, P. R. Tm-builder: Um construtor de ontologias baseado em topic maps. 2003. Citado 2 vezes nas páginas 24 e 33.

LUCIA, A. D. et al. Improving artefact quality management in advanced artefact management system with distributed inspection. *Software, IET*, v. 5, n. 6, p. 510–527, 2011. ISSN 1751-8806. Citado 2 vezes nas páginas 46 e 48.

MCGUINNESS, D. L.; HARMELEN, F. van. Owl web ontology language overview. 2004. Citado na página 34.

MISHRA, D.; MISHRA, A. Simplified software inspection process in compliance with international standards. *Computer Standards & Interfaces*, v. 31, n. 4, p. 763 – 771, 2009. ISSN 0920-5489. Disponível em: <http://www.sciencedirect.com/science/article-/pii/S0920548908001177>. Citado 2 vezes nas páginas 45 e 47.

MISHRA, D.; MISHRA, A. A global software inspection process for distributed software development. *J. UCS*, v. 18, n. 19, p. 2731–2746, 2012. Citado 2 vezes nas páginas 46 e 47.

MOTIK, B. et al. Owl 2 web ontology language: Profiles. *W3C recommendation*, v. 27, p. 61, 2009. Citado 2 vezes nas páginas 34 e 35.

NODLER, J.; NEUKIRCHEN, H.; GRABOWSKI, J. A flexible framework for quality assurance of software artefacts with applications to java, uml, and ttcn-3 test specifications. In: *Software Testing Verification and Validation, 2009. ICST '09. International Conference on*. [S.l.: s.n.], 2009. p. 101–110. Citado 2 vezes nas páginas 47 e 48.

O'CONNOR, M. et al. Supporting rule system interoperability on the semantic web with swrl. Citado na página 38.

OMG Object Management Group. *OMG Systems Modeling Language (OMG SysML^{TM}) Versão 1.3*. 2010. Disponível em: <http://www.omg.org/spec/SysML/1.3/>. Citado na página 39.

OWL API. Disponível em: <http://owlapi.sourceforge.net/>. Citado na página 42.

PAULA FILHO, W. d. P. *Engenharia de software :fundamentos, metodos e padroes*. 3 ed.. ed. Rio de Janeiro: LTC, 2009. 1248 p. p. Citado na página 23.

PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. Semantics and complexity of sparql. In: *The Semantic Web-ISWC 2006*. [S.l.]: Springer, 2006. p. 30–43. Citado na página 36.

PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF*. 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Citado 2 vezes nas páginas 36 e 37.

SILVA, J. P. S. da et al. Um sistema para inspeções de garantia da qualidade baseado em ontologias e agentes. *Revista de Informática Teórica e Aplicada*, XX, n. 3, p. 13–30, 2013. Citado 2 vezes nas páginas 47 e 48.

Stanford Center for Biomedical Informatics Research. *Protégé*. 2014. Disponível em: <http://protege.stanford.edu/>. Citado na página 71.

W3C. *Semantic Web*. 2013. Disponível em: <http://www.w3.org/standards-/semanticweb/ontology>. Citado na página 32.

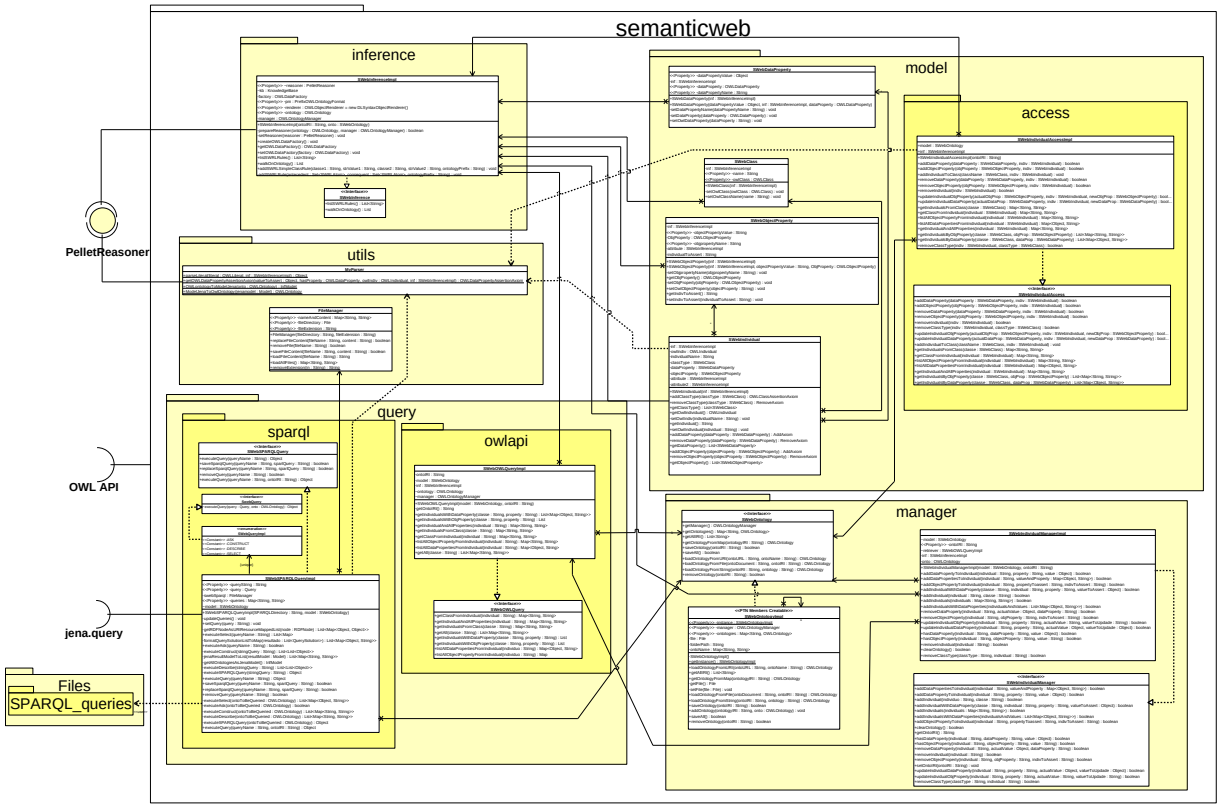
W3C Owl Working Group et al. Owl 2 web ontology language document overview. *W3C Recommendation*, v. 27, p. 1205–1214, 2009. Citado 2 vezes nas páginas 34 e 35.

WAGNER FILHO, F.; LÓSCIO, B. F. Web semântica: Conceitos e tecnologias. Disponível em: <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso-/mc9.pdf>. Citado na página 32.

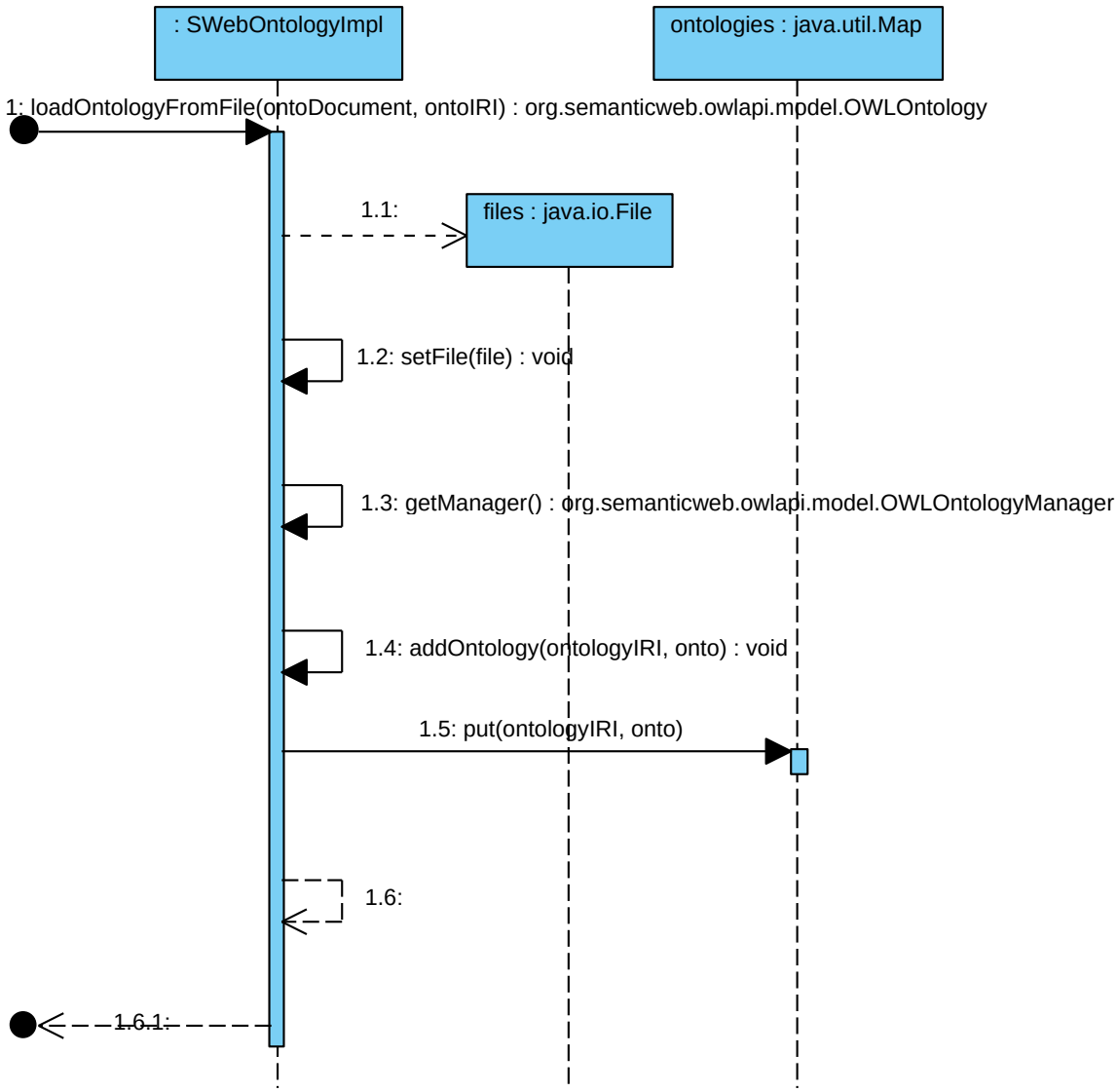
WANG, M.-H.; LEE, C.-S. An intelligent ppqa web services for cmmi assessment. In: *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on*. [S.l.: s.n.], 2008. v. 1, p. 229–234. Citado 2 vezes nas páginas 47 e 48.

Apêndices

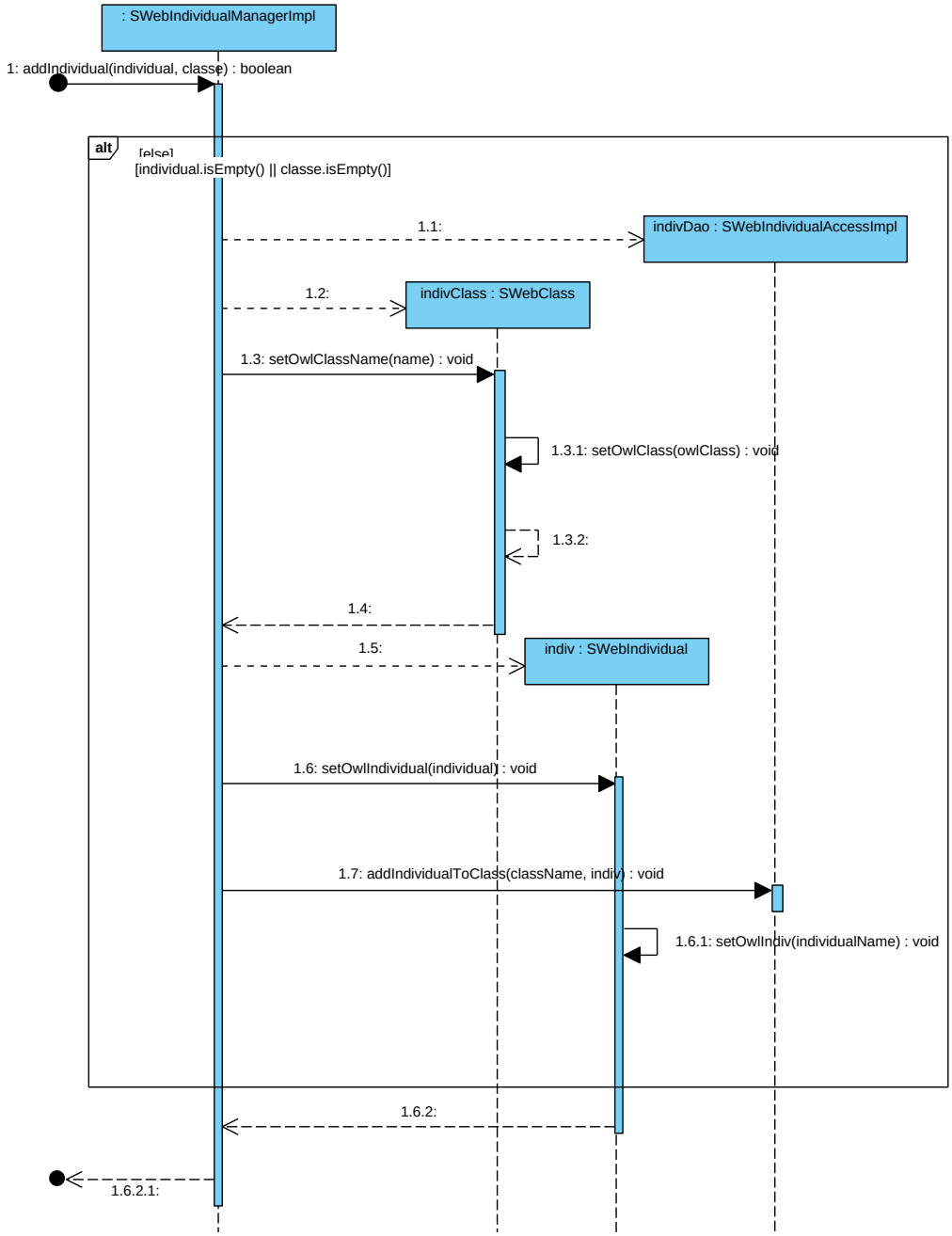
APÊNDICE A – Diagrama de Classes



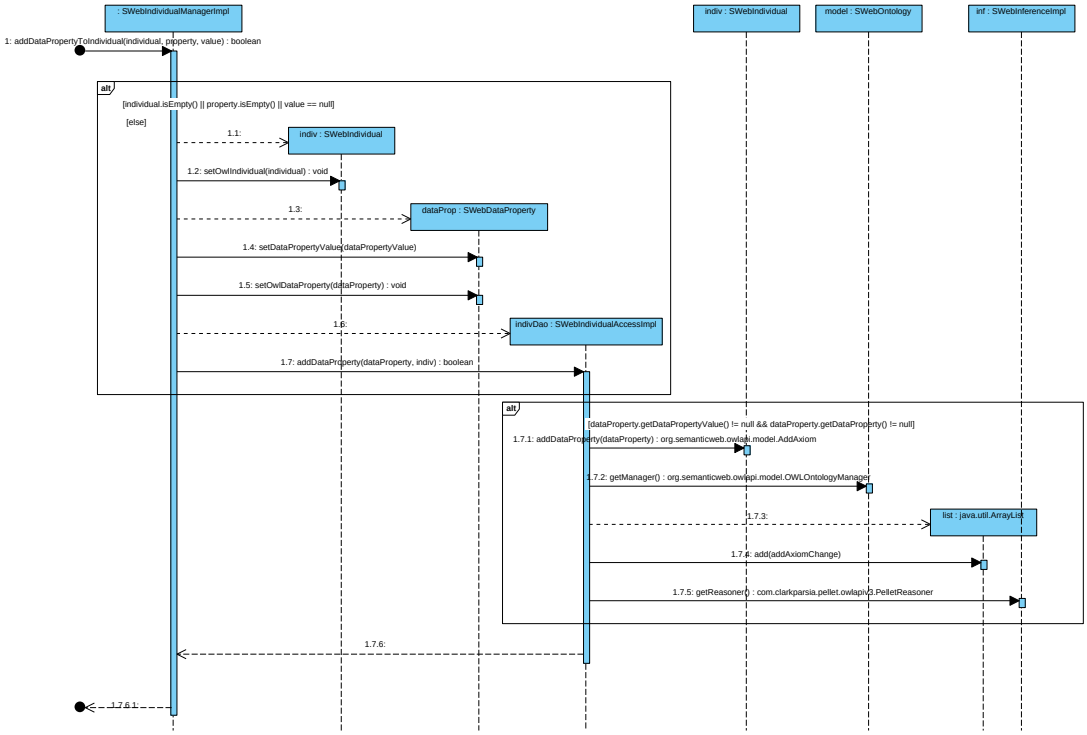
APÊNDICE B – Diagrama de Sequência - Carregar Ontologia



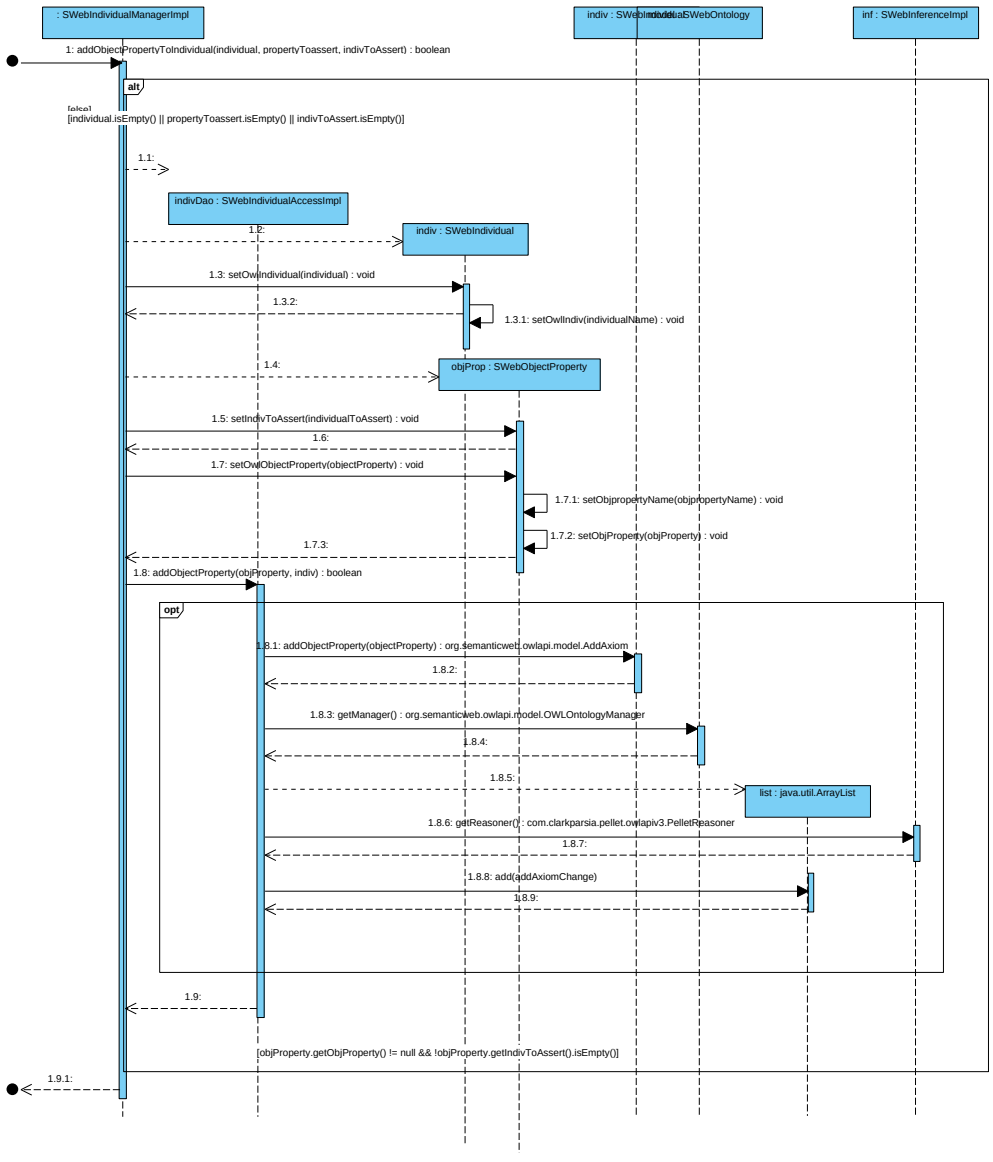
APÊNDICE C – Diagrama de Sequência - Adicionar Indivíduo



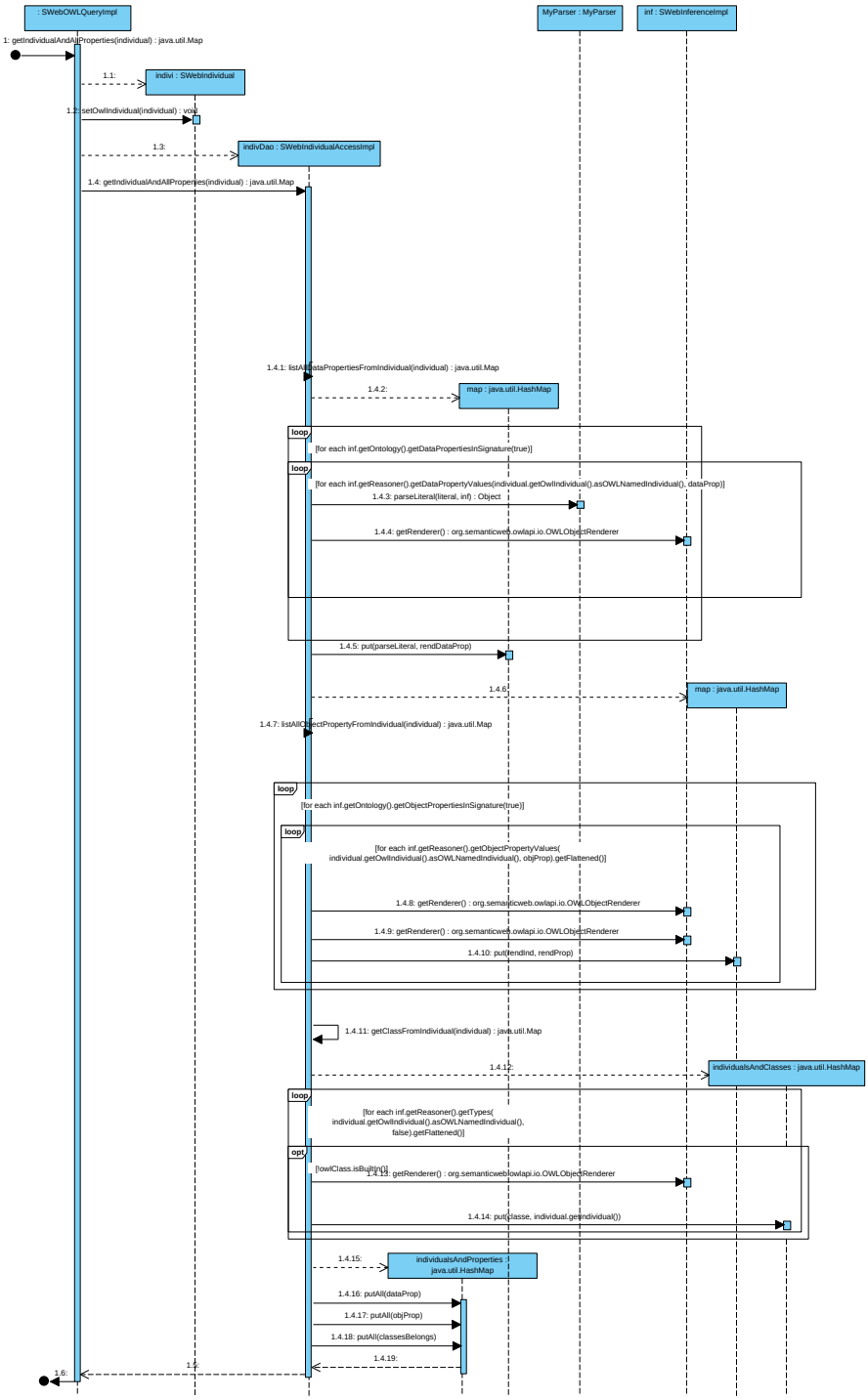
APÊNDICE D – Diagrama de Sequência - Adicionar Propriedade Tipo Dado ao Indivíduo



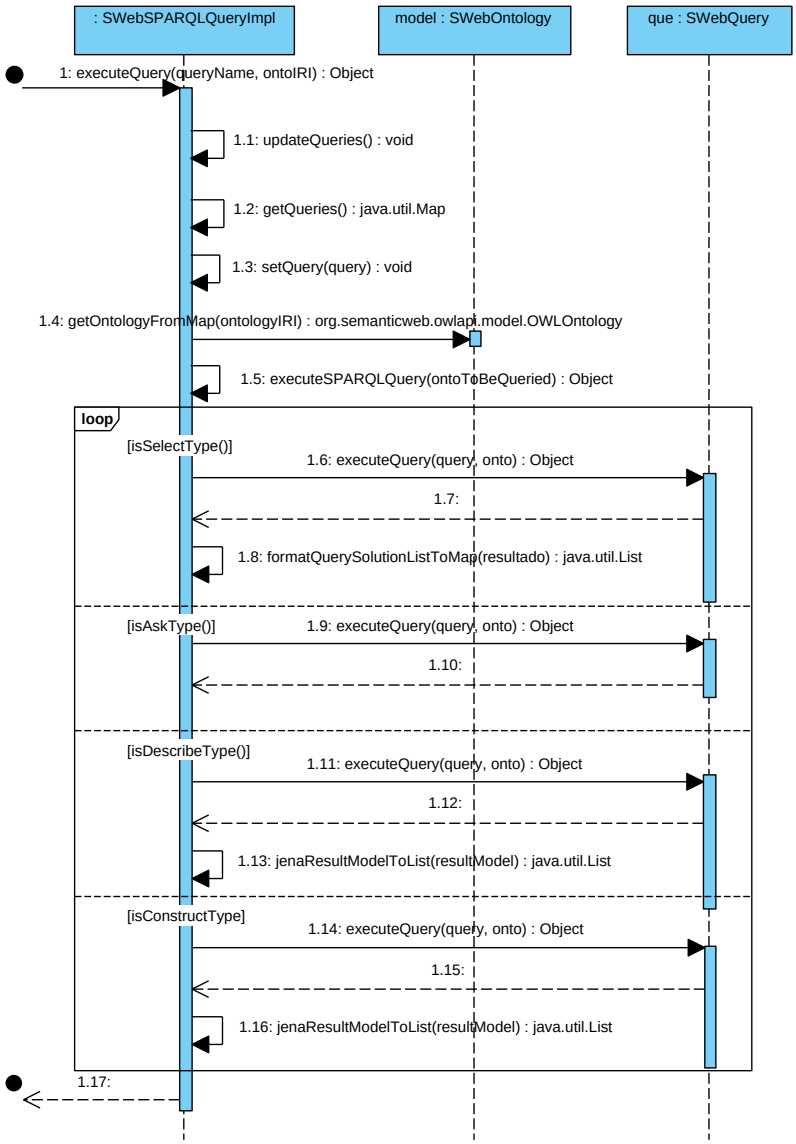
APÊNDICE E – Diagrama de Sequência - Adicionar Propriedade Tipo Objeto ao Indivíduo



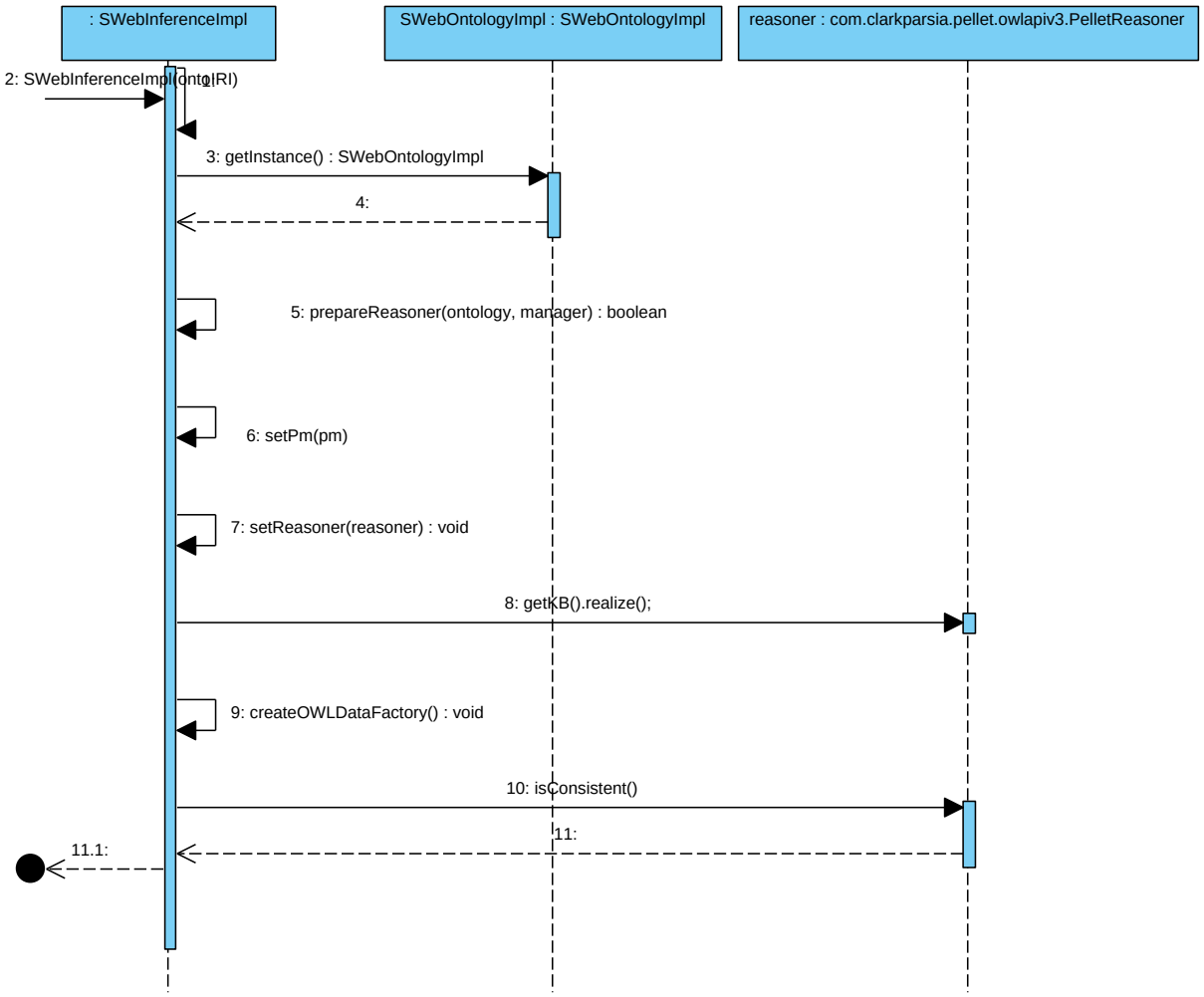
APÊNDICE F – Diagrama de Sequência - Recuperar Indivíduos e Suas Propriedades


: SWebOWLQueryImpl
MyParser : MyParser
inf : SWebInferenceImpl
1: getIndividualAndAllProperties(individual) : java.util.Map
1.1:
indivi : SWebIndividual
1.2: setOwlIndividual(individual) : void
1.3:
indivDao : SWebIndividualAccessImpl
1.4: getIndividualAndAllProperties(individual) : java.util.Map
1.4.1: listAllDataPropertiesFromIndividual(individual) : java.util.Map
1.4.2:
map : java.util.HashMap
loop
[for each inf.getOntology().getDataPropertiesInSignature(true)]
loop
[for each inf.getReasoner().getDataPropertyValues(individual.getOwlIndividual().asOWLNamedIndividual(), dataProp)]
1.4.3: parseLiteral(literal, inf) : Object
1.4.4: getRenderer() : org.semanticweb.owlapi.io.OWLObjectRenderer
1.4.5: put(parseLiteral, rendDataProp)
1.4.6:
map : java.util.HashMap
1.4.7: listAllObjectPropertyFromIndividual(individual) : java.util.Map
loop
[for each inf.getOntology().getObjectPropertiesInSignature(true)]
loop
[for each inf.getReasoner().getObjectPropertyValues(
individual.getOwlIndividual().asOWLNamedIndividual(), objProp).getFlattened()]
1.4.8: getRenderer() : org.semanticweb.owlapi.io.OWLObjectRenderer
1.4.9: getRenderer() : org.semanticweb.owlapi.io.OWLObjectRenderer
1.4.10: put(rendInd, rendProp)
1.4.11: getClassFromIndividual(individual) : java.util.Map
1.4.12:
individualsAndClasses : java.util.HashMap
loop
[for each inf.getReasoner().getTypes(
individual.getOwlIndividual().asOWLNamedIndividual(),
false).getFlattened()]
opt
[!owlClass.isBuiltIn()]
1.4.13: getRenderer() : org.semanticweb.owlapi.io.OWLObjectRenderer
1.4.14: put(classe, individual.getIndividual())
1.4.15:
individualsAndProperties :
java.util.HashMap
1.4.16: putAll(dataProp)
1.4.17: putAll(objProp)
1.4.18: putAll(classesBelongs)
1.4.19:
1.5:
1.6:

APÊNDICE G – Diagrama de Sequência - Executar Consulta SPARQL



APÊNDICE H – Diagrama de Sequência - Raciocinar Ontologia



APÊNDICE I – Diagrama de Sequência - Listar Regras SWRL